

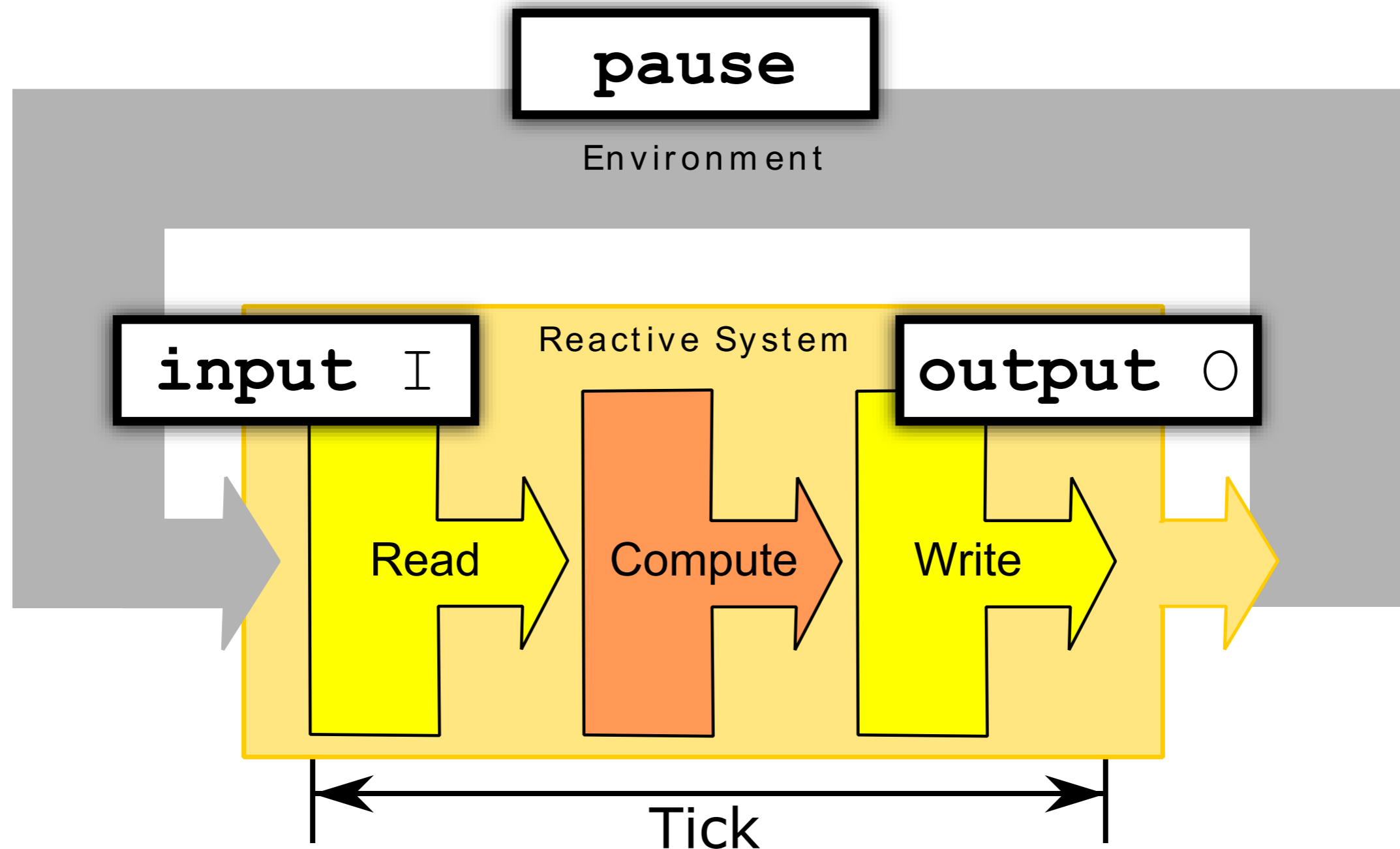
# Sequential Constructiveness, SCL and SCCharts

Incorporating synchrony in  
conventional languages

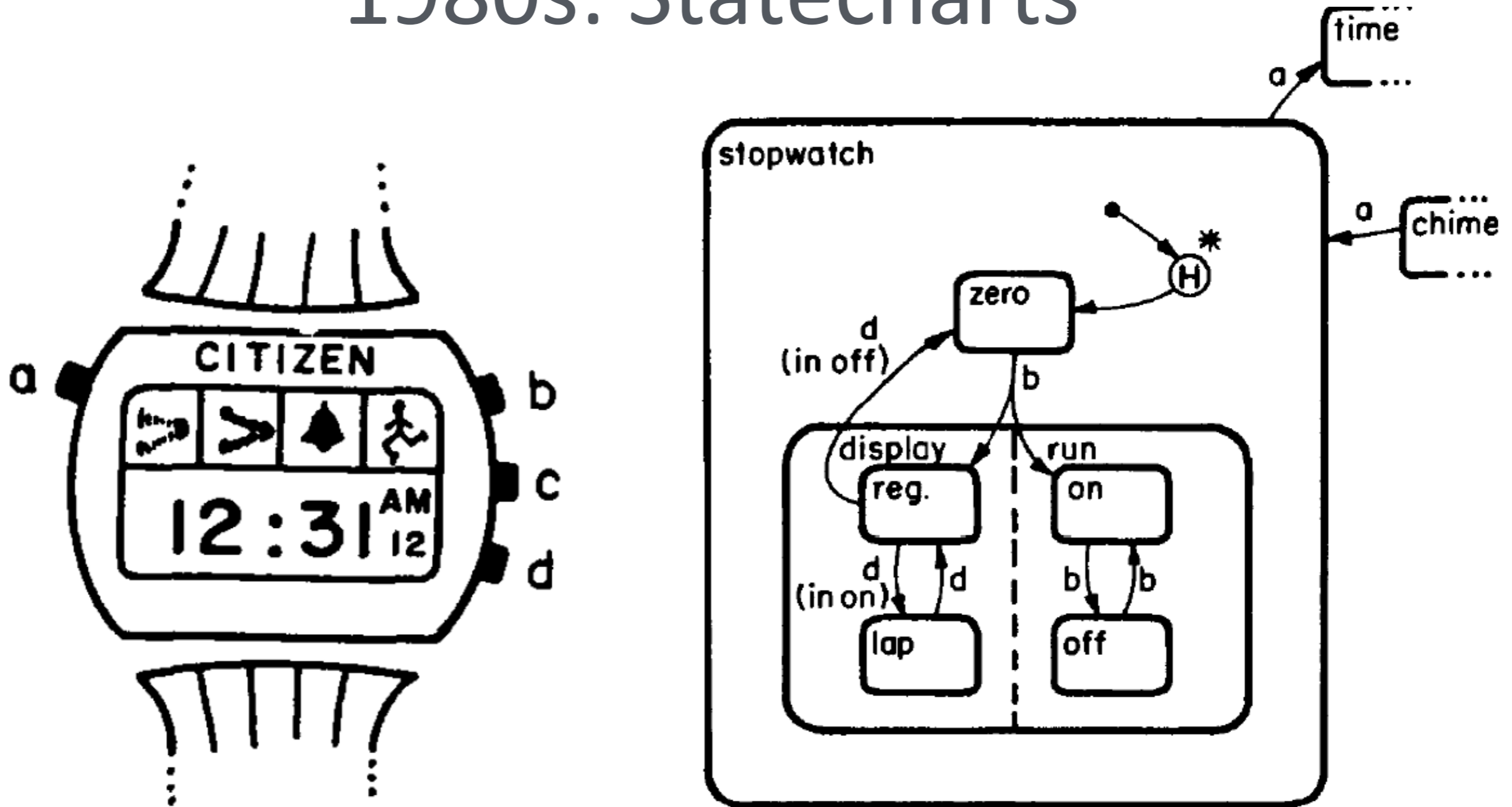
Reinhard von Hanxleden (U Kiel)

14 March 2018, Collège de France

# Reactive Systems



# 1980s: Statecharts



[David Harel,  
*Statecharts: A Visual Formalism for Complex Systems*,  
*Science of Computer Programming*, 1987]

# 1990s: Many Statecharts

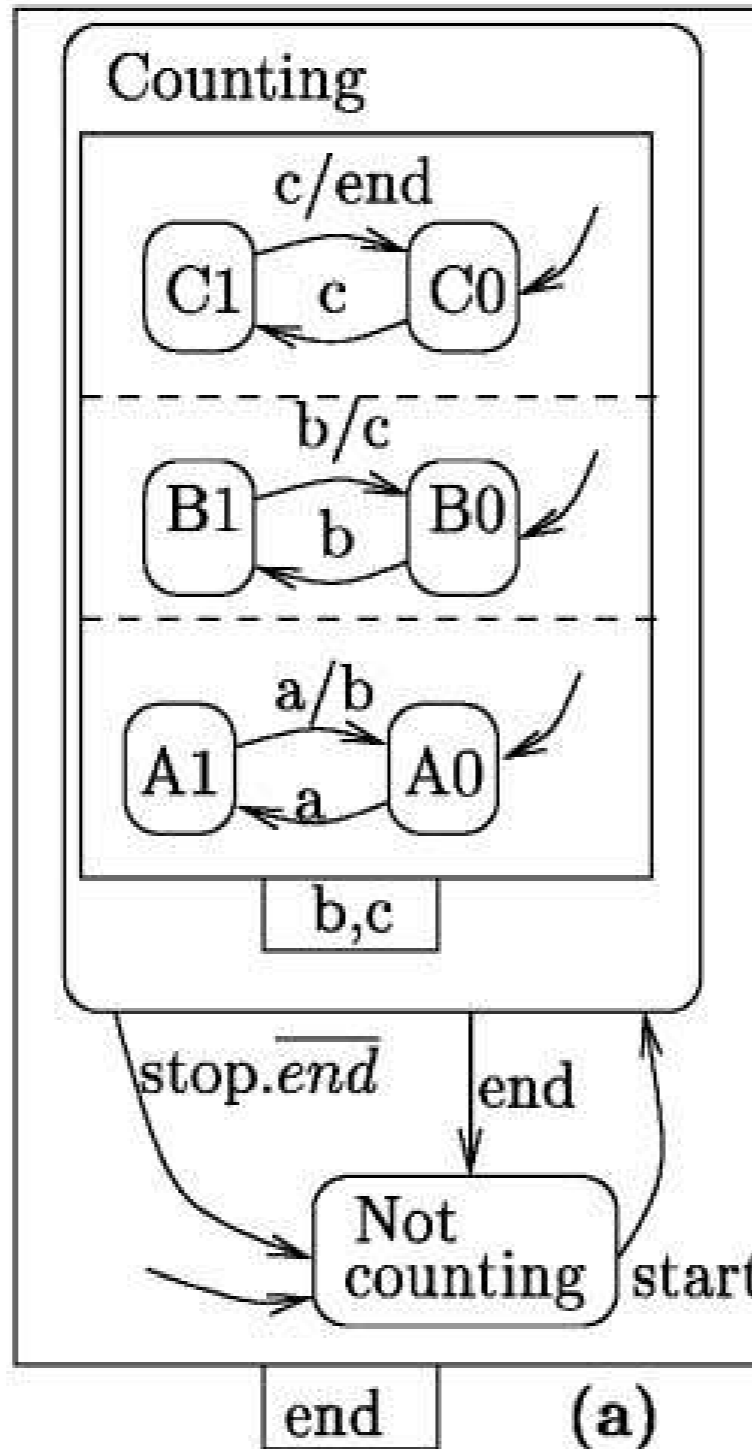
	Variant									
	1	2	3	4	5	6	7	8	9	10
graphical / textual	g	g	g	g	g	g	g	g/tg/t	g	g
negated trigger event	+	+	+	+	+	+	+	+	+	-
timeout event	+	-	-	-	-	-	-	+	+	+ <sup>9</sup>
timed transition	-	-	-	-	-	-	-	-	-	-
disjunction of trigger events	-	+	+	+	+	- <sup>13</sup>	- <sup>13</sup>	+	+	-

Statecharts Feature	Variant Number																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
graphical / textual	g	g	g	g	g	g	g	g/tg/t	g	g	g/t	g	g	g	g	g	g	g	g	g	g
negated trigger event	+	+	+	+	+	+	+	+	-	+	-	+	+	-	-	-	-	-	+	+	+
timeout event	+	-	-	-	-	-	-	-	+	+	-	-	-	+	+	+	+	+	+	+	+
timed transition	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+
disjunction of trigger events	-	+	+	+	+	- <sup>13</sup>	- <sup>13</sup>	+	+	-	-	-	+	+	-	-	-	-	-	-	+
trigger condition	+	+	-	-	-	- <sup>13</sup>	- <sup>13</sup>	+	-	+	-	-	-	+	-	-	-	-	-	+	+
state reference	+	+	-	-	-	- <sup>13</sup>	- <sup>13</sup>	+	-	+	-	-	-	+	-	-	-	-	-	-	-
assignment to variable	+	+	-	-	-	- <sup>13</sup>	- <sup>13</sup>	-	-	+	-	-	-	+	-	-	-	-	-	+	+
inter-level transition	+	+	- <sup>14</sup>	- <sup>14</sup>	- <sup>14</sup>	+	+	+	+	+	-	-	+	+	+	+	+	+	+	-	-
history mechanism	+	+	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-
operational/denotatio.	-	o	o	o	o	o	o	d	d	o	o	d	o	o	d	d	d	d	o	o	?
compositional	-	-	-	-	-	-	-	+	+	+	+	-	+	+	+	+	+	+	+	+	+
synchrony hypothesis	+	+	+	+	+	+	+	+	+	+	+	+	+	-	-	-	-	-	-	- <sup>8</sup>	- <sup>8</sup>
deterministic	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
interleav./true concurr.	i	i	i	i	i	i	i	i	i	i	i	i	i	i	t	t	t	t	t	<sup>15</sup>	<sup>15</sup>
discrete/contin. time	d	d	d	d	d	d	d	d	d	d	d	d	d	d	c	c	c	c	c	c	c
globally consistent	-	-	-	+	+	+	+	- <sup>13</sup>	+	+	- <sup>5</sup>	- <sup>5</sup>	-	- <sup>5</sup>	- <sup>5</sup>	- <sup>5</sup>	- <sup>5</sup>	- <sup>5</sup>	-	-	-
causal	+	+	+	+	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
instantaneous state	?	-	-	-	-	-	-	-	-	+	-	+	-	-	-	-	-	-	+	+	+
finite transition no.	?	+	+	+	+	+	+	+	+	-	+	-	+	+	+	+	+	+	+	-	+
priorities	-	-	- <sup>14</sup>	- <sup>14</sup>	- <sup>14</sup>	- <sup>10</sup>	- <sup>10</sup>	-	-	-	-	-	-	+	-	+	-	+	-	-	+
non-preempt. interrupt	?	-	- <sup>14</sup>	- <sup>14</sup>	- <sup>14</sup>	-	-	-	-	?	+	-	-	-	-	-	-	-	+	+	?
preemptive interrupt	?	+	- <sup>14</sup>	- <sup>14</sup>	- <sup>14</sup>	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
distinct. int./ext. event	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
local event	-	-	-	-	-	-	-	-	+	+	+	+	+	-	-	-	-	-	-	-	+
discrete/contin. event	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	c	c	c	c	d	d

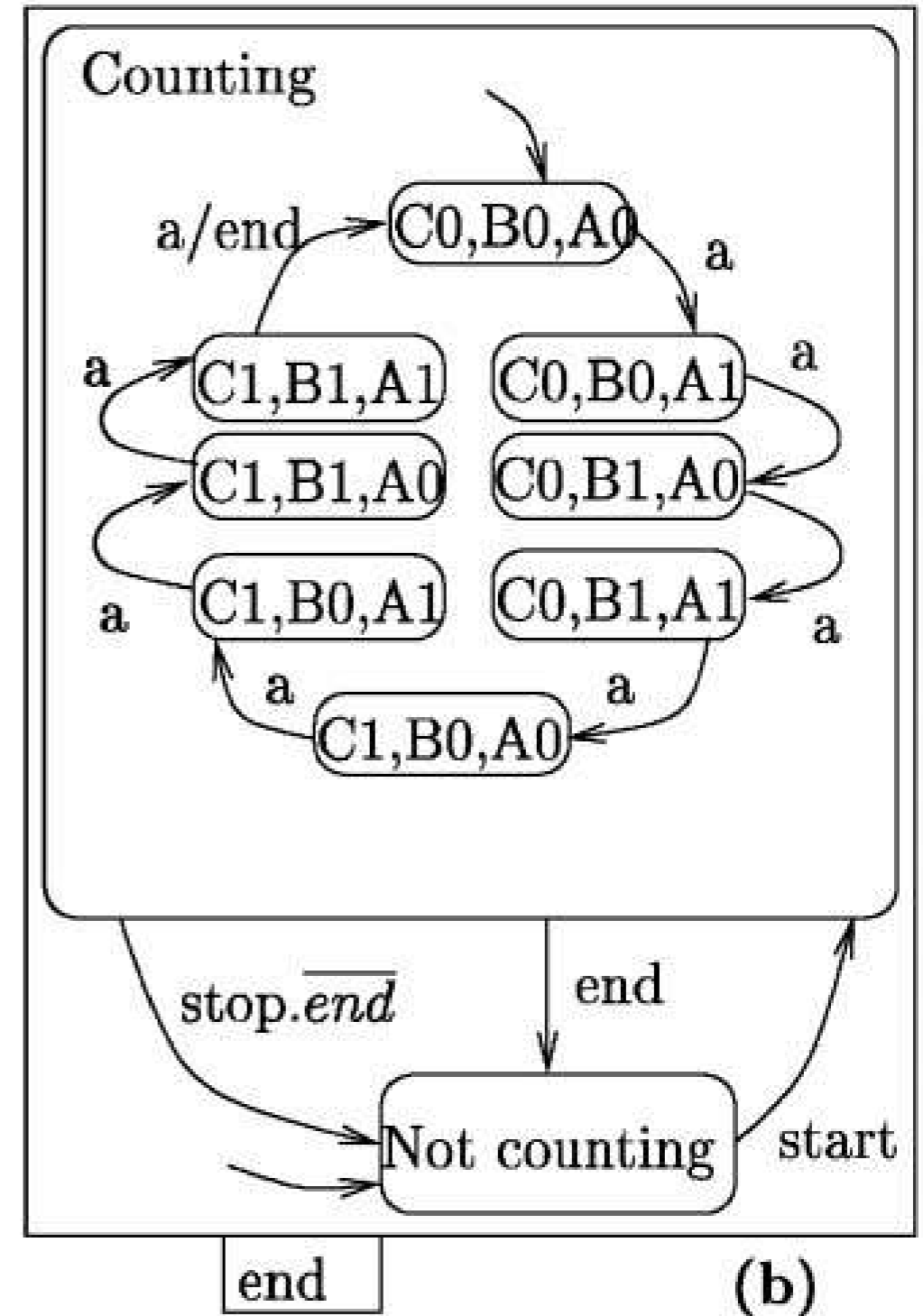
[Michael von der Beeck, *A Comparison of Statecharts Variants*, Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863, 1994]

# 1991: Argos

Main1 (a, start, stop) ()

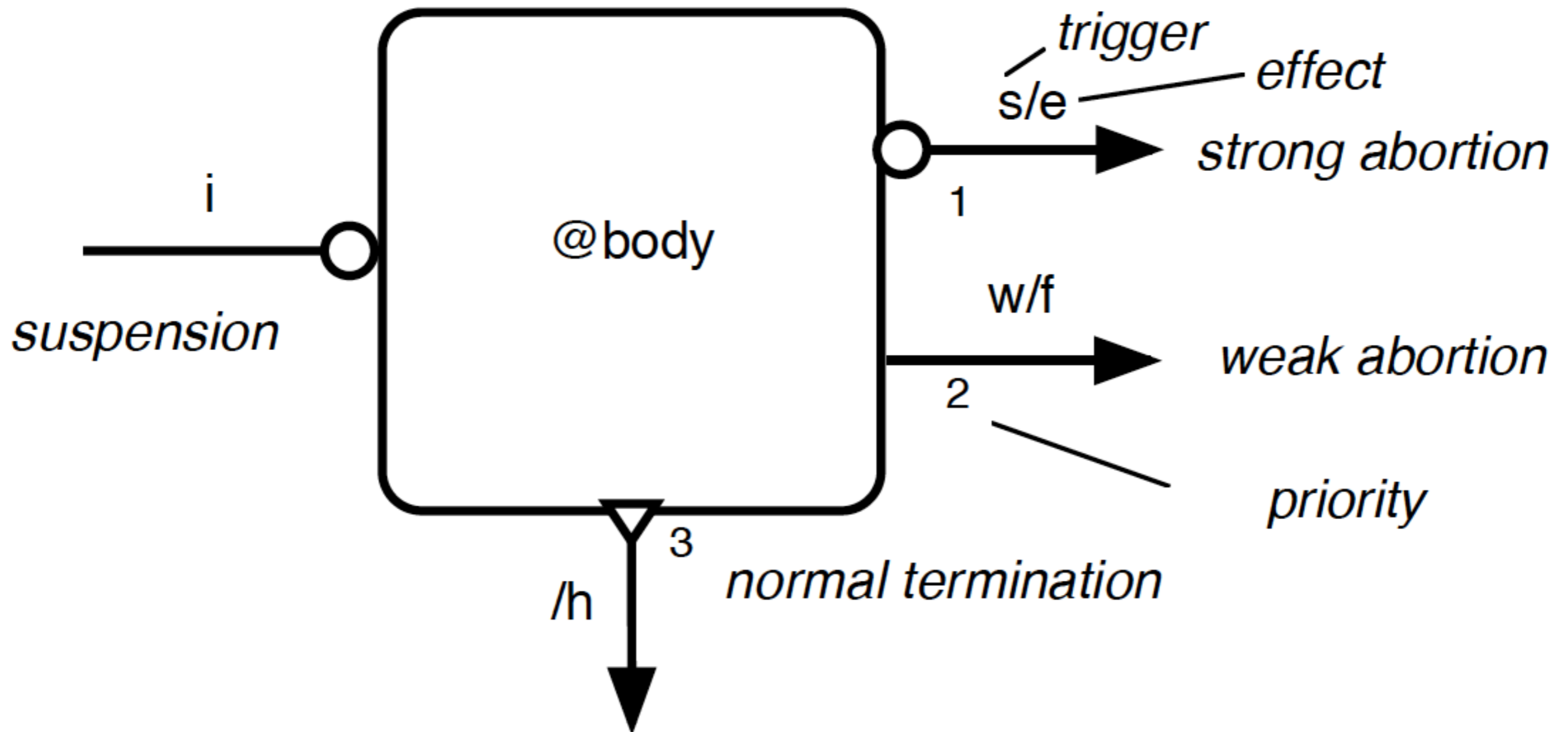


Main2 (a, start, stop) ()



[Florence Maraninchi, *The Argos language: Graphical Representation of Automata and Description of Reactive Systems*, IEEE Workshop on Visual Languages, Kobe, Japan, 1991]

# 1995: SyncCharts, a.k.a. Safe State Machines



[Charles André,  
*SyncCharts: A Visual Representation of Reactive Behaviors*,  
Research Report 95-52, I3S, Sophia Antipolis, 1995]

# SCCharts – Motivation

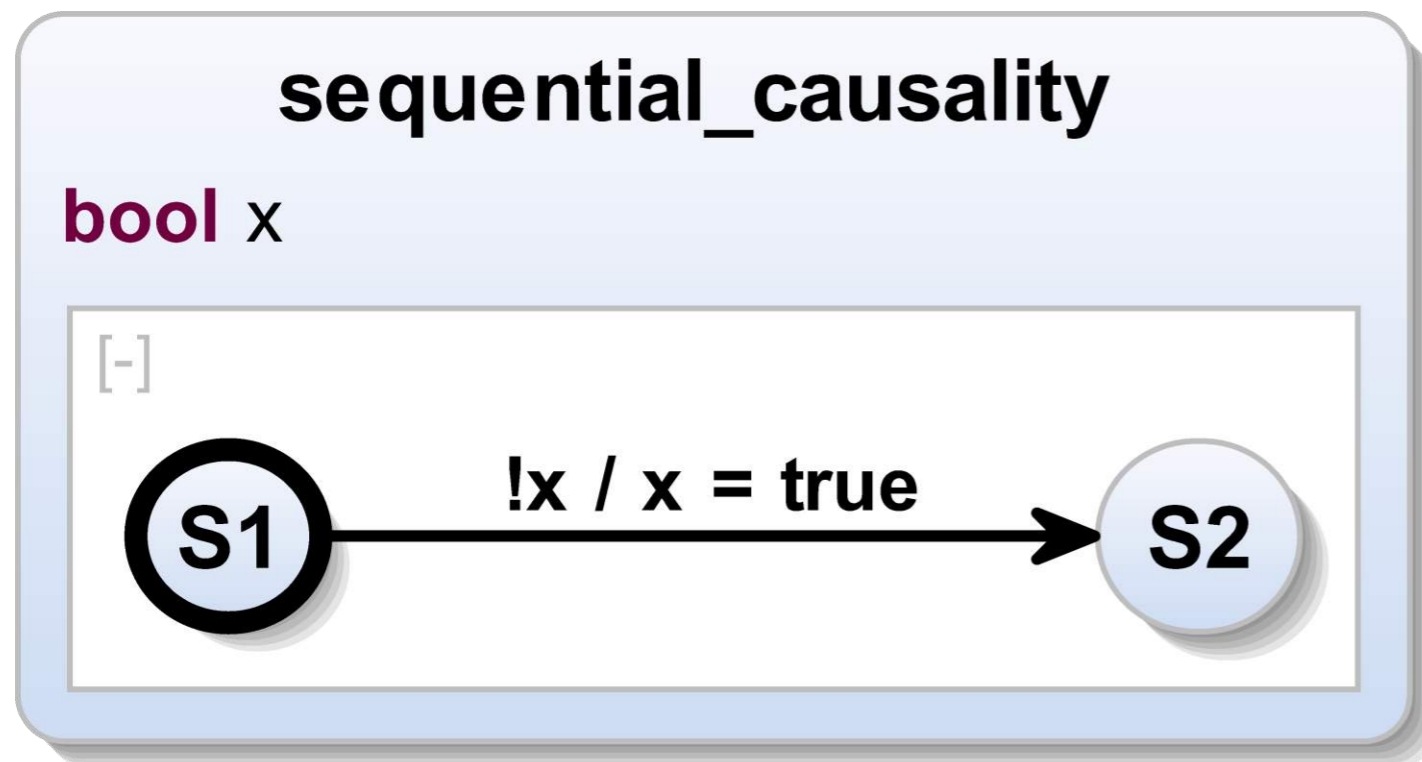
# Limitations of Strict Synchrony

```
if (!x) {  
    x = true;  
}
```

Good C



# Limitations of Strict Synchrony



Bad SyncChart

Good SCChart

# Limitations of Strict Synchrony

```
present x else  
  emit x  
end
```

Bad Esterel

Good SCEst

# SCCharts – Motivation

Preserve nice properties of synchronous programming

- Determinacy
- Sound semantic basis
- Efficient synthesis

Reduce the pain

- Make it easy to adapt for mainstream programmer
- Reject only models where determinacy is compromised

# Model of Computation

# Sequential Constructiveness

Sequential control flow overrides „write before read“

Writes visible only to reads that are

1. sequential successors or
2. concurrent

[v. Hanxleden, Mendler, et al.,

*Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation,*

ACM TECS '14]

# SCCharts – MoC

It's all about scheduling variable accesses within reaction ...

- Sequential accesses to  $x$ : unconstrained

- Concurrent accesses to  $x$  („iur protocol“):

init  $\Rightarrow$  updates  $\Rightarrow$  reads  
 $x = 1 \dots x += 2 \dots x += 5 \dots y = x \dots z = x$

- Concurrent accesses may lead to causality cycles – compiler must reject those

# SCCharts – MoC

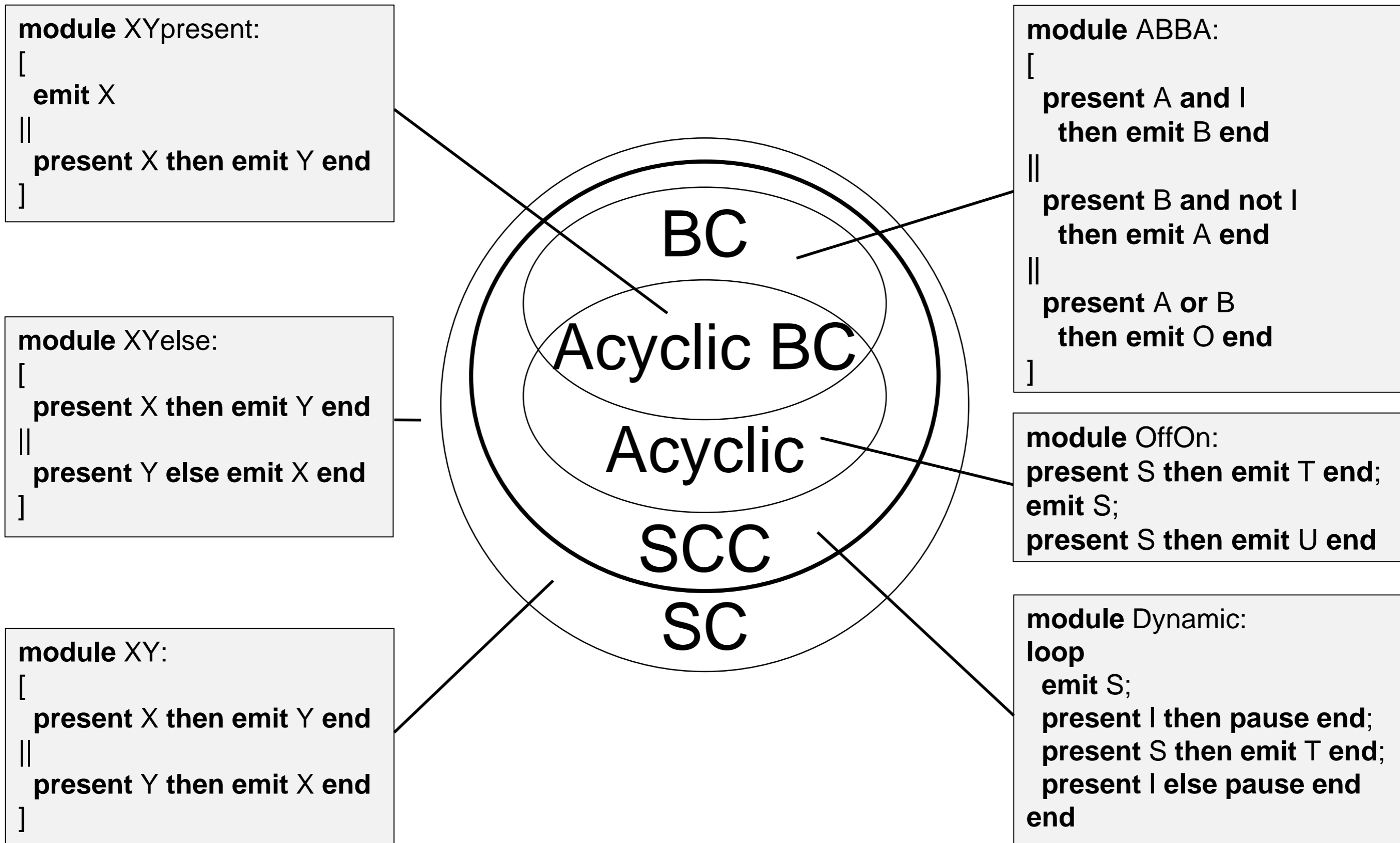
SCChart / SCEst program is ...

... **SC**, „is sequentially constructive,“ if

1. there exist runs obeying iur protocol
2. all such runs produce same result

... **SCC**, „corresponds to sequentially constructive circuit,“ if it is SC and does not „speculate“

# Program Classes





# SCCharts – The Language(s)

Interface  
declaration

ABRO

**input bool** A,B,R  
**output bool** O = false

Initialization

Interface  
declaration

ABRO

**input bool** A,B,R  
**output bool** O = false

Initialization

[-]

ABthenO

Superstate

Interface declaration

Initialization

Superstate

Region



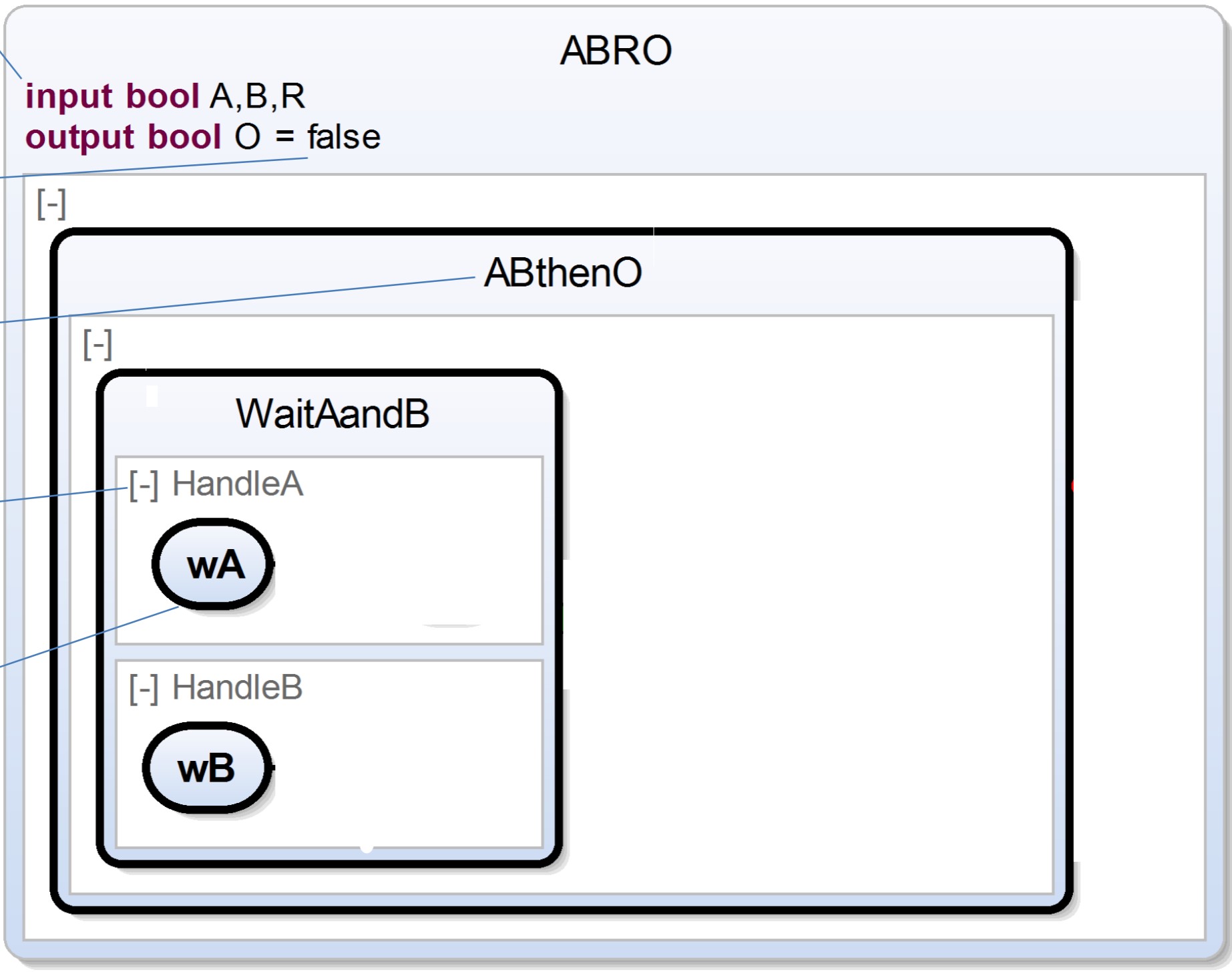
Interface declaration

Initialization

Superstate

Region

Initial state



Interface declaration

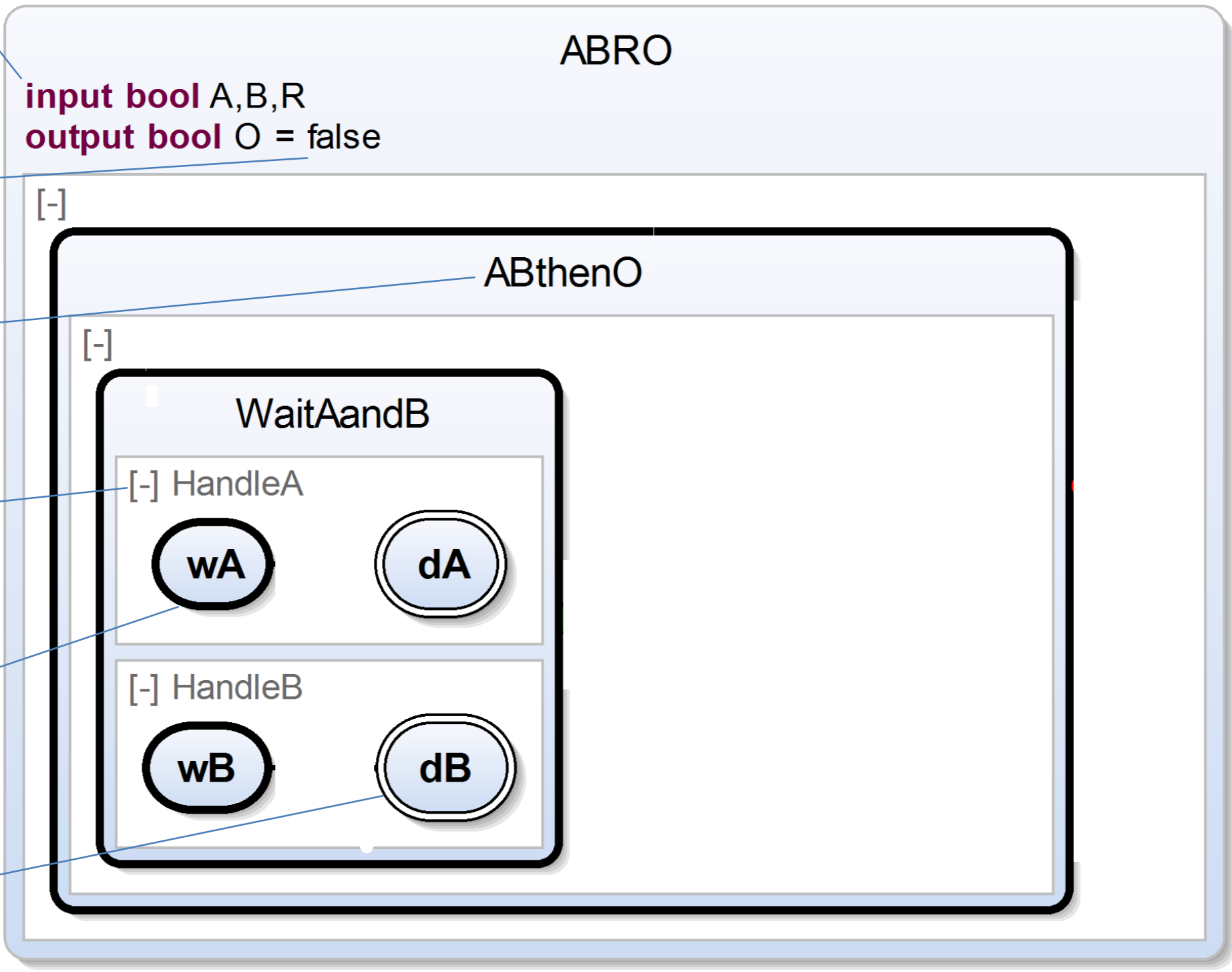
Initialization

Superstate

Region

Initial state

Final state



Interface declaration

**input bool** A,B,R  
**output bool** O = false

Delayed Transition  
(+ Trigger)

Initialization

[-]

ABthenO

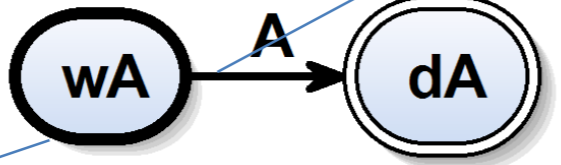
Superstate

[-]

WaitAandB

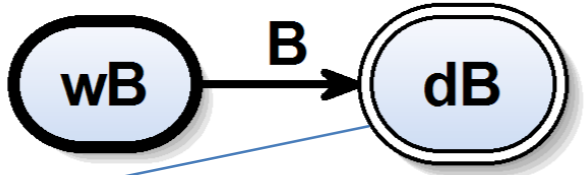
Region

[-] HandleA

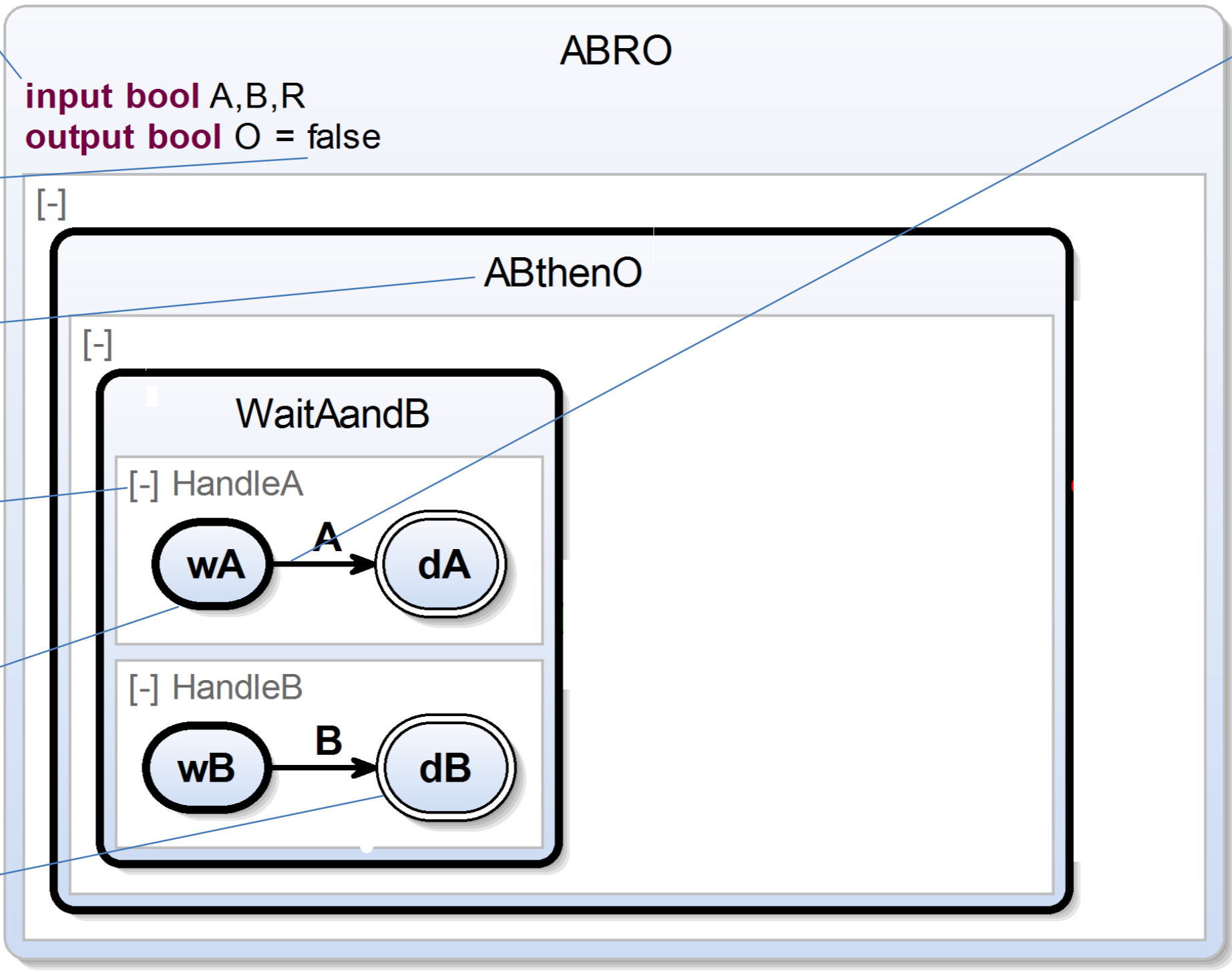


Initial state

[-] HandleB



Final state



Interface declaration

**input bool** A,B,R  
**output bool** O = false

Initialization

[-]

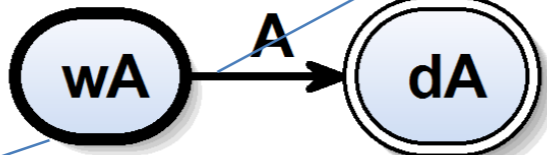
Superstate

ABthenO

[-]

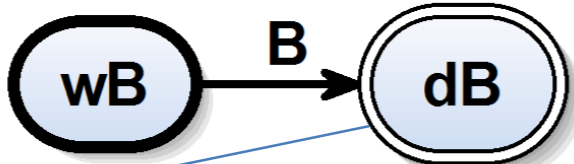
Region

[-] HandleA

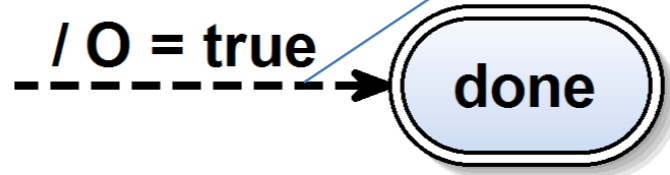


Initial state

[-] HandleB



Final state



Delayed Transition (+ Trigger)

Immediate transition (+ Effect)



Interface declaration

**input bool** A,B,R  
**output bool** O = false

Initialization

[-]

Superstate

[-]

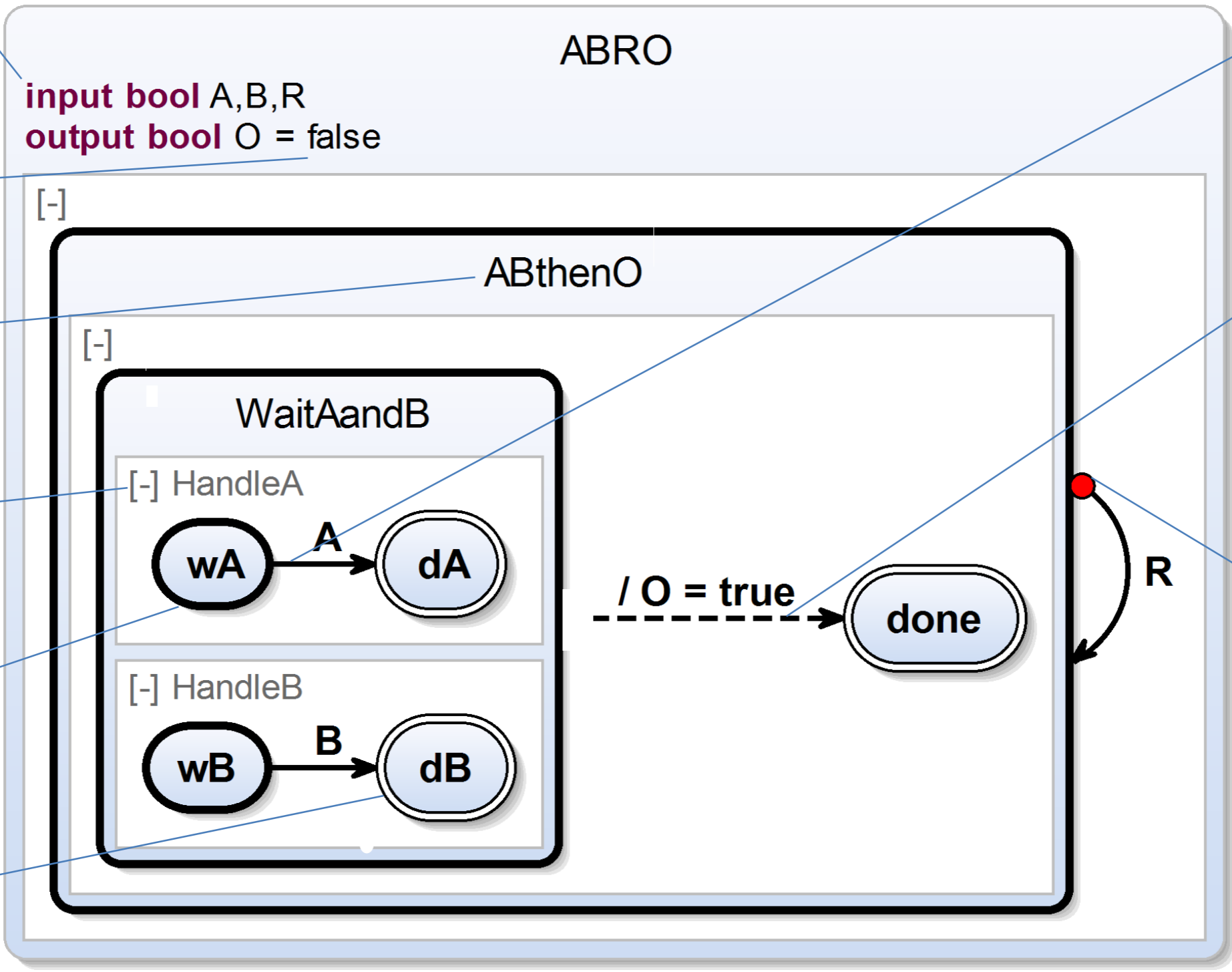
Region

[-] HandleA

Initial state

[-] HandleB

Final state



ABRO

ABthenO

WaitAandB

done

/ O = true

R

Delayed Transition (+ Trigger)

Immediate transition (+ Effect)

Strong abort

Interface declaration

**input bool** A,B,R  
**output bool** O = false

Initialization

[-]

Superstate

ABthenO

[-]

Region

[-] HandleA

Initial state

[-] HandleB

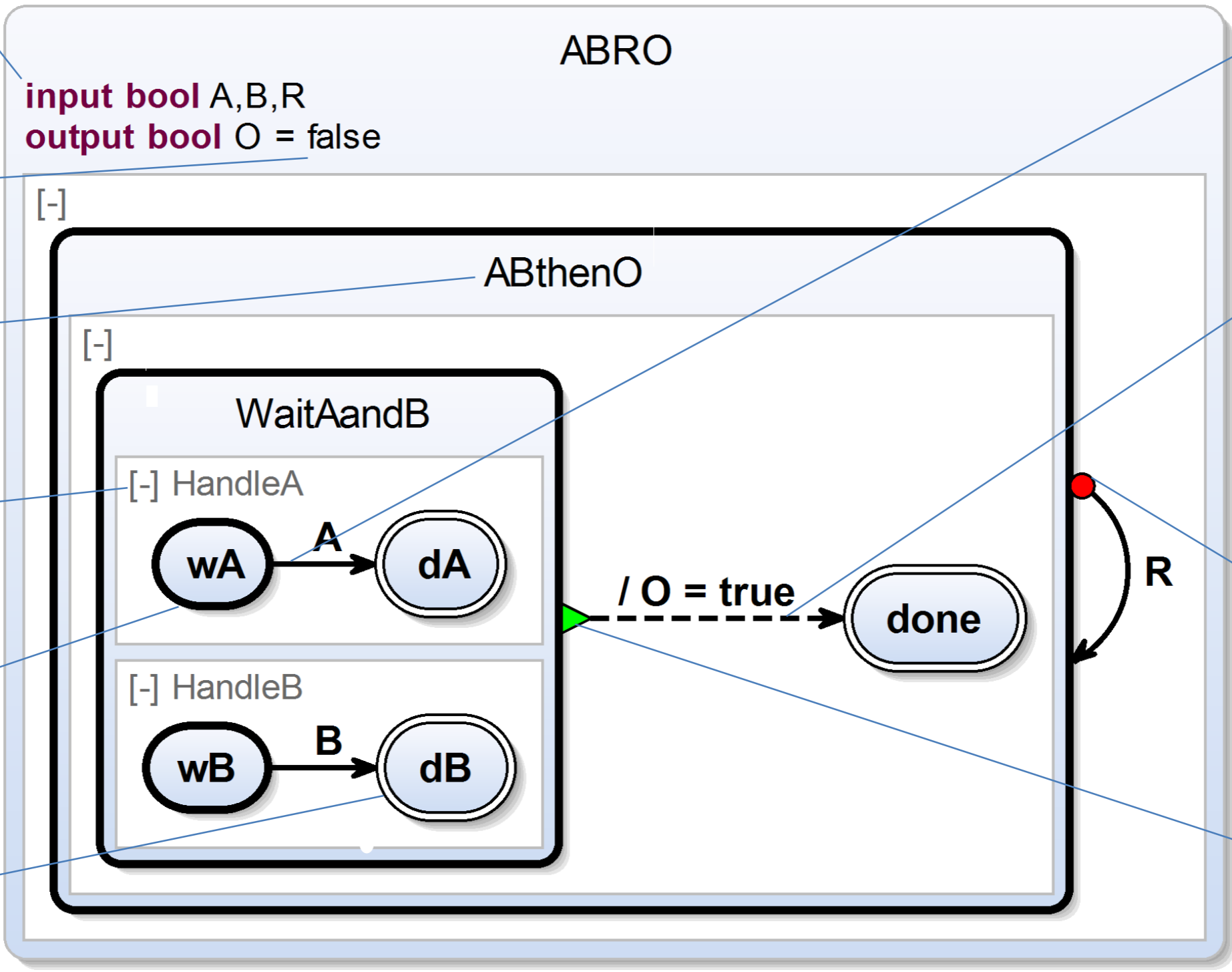
Final state

Delayed Transition (+ Trigger)


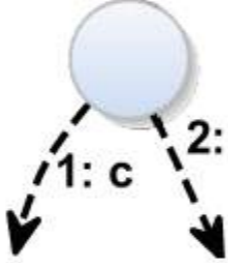
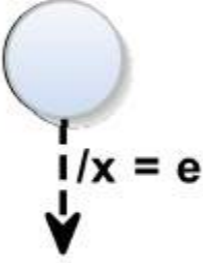
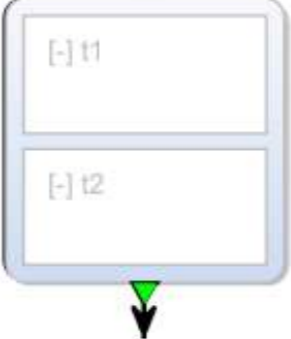

Immediate transition (+ Effect)


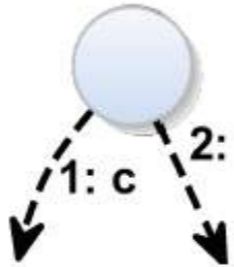
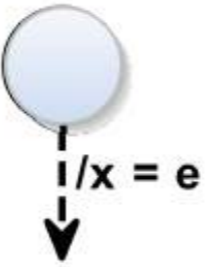
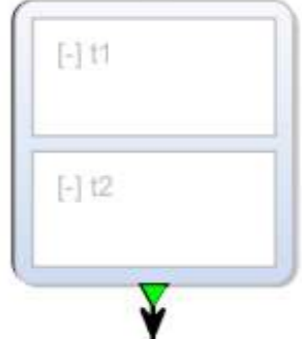

Strong abort

Termination

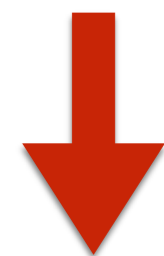


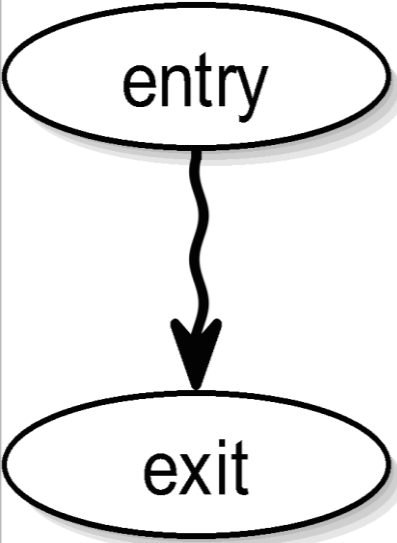
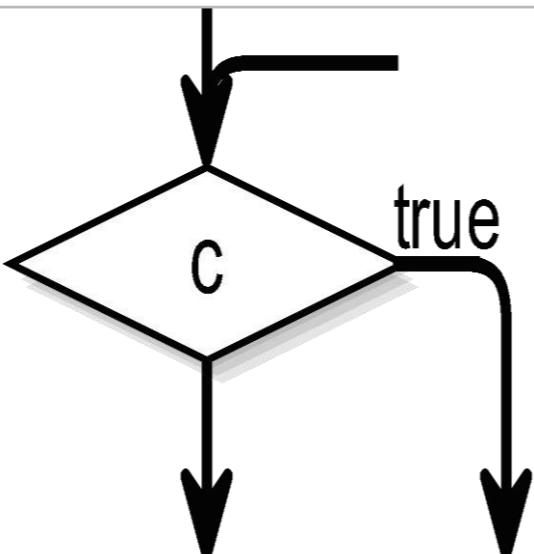
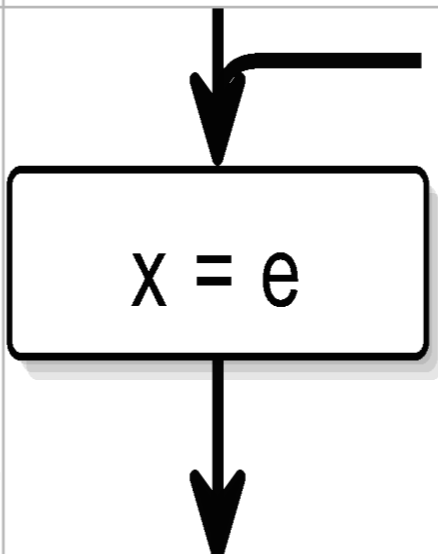
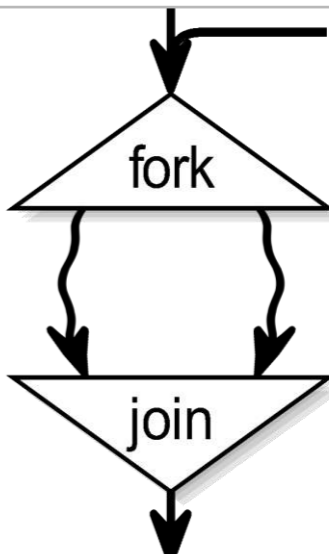
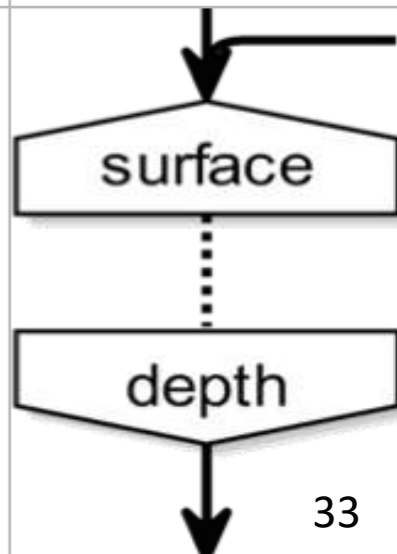
# SCChart Building Blocks

	Region	Trigger	Effect	Superstate	State
<b>Normalized Core SCCharts</b>					

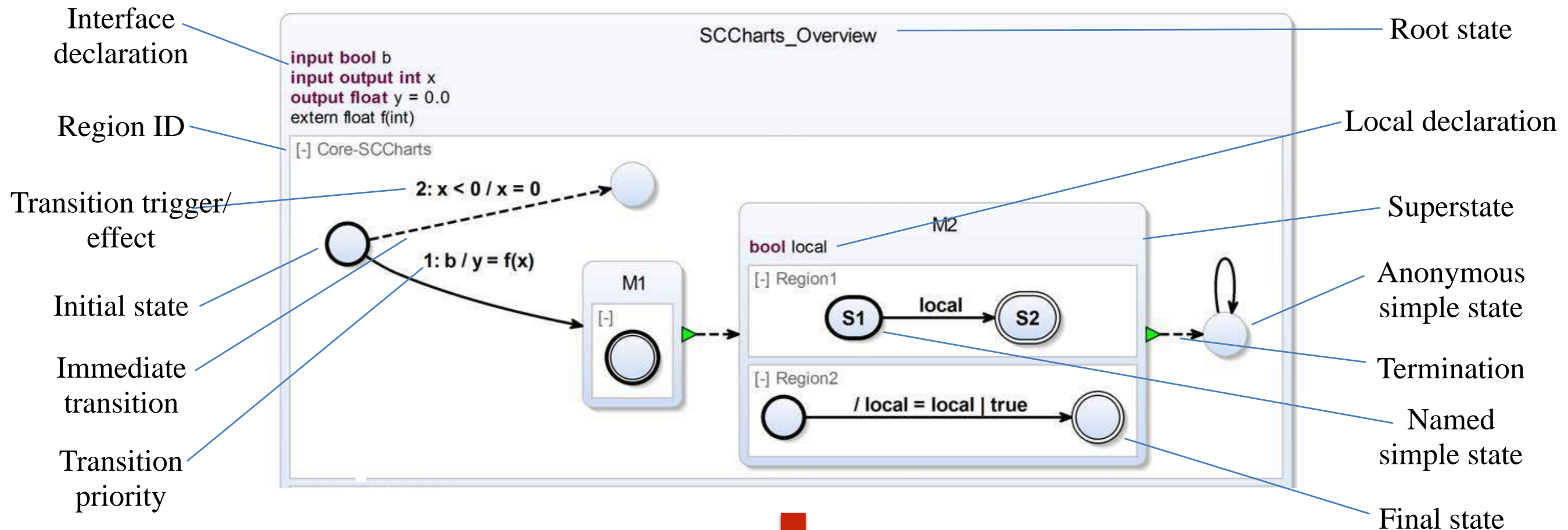
	Region	Trigger	Effect	Superstate	State
Normalized Core SCCharts					

### M2M Mappings



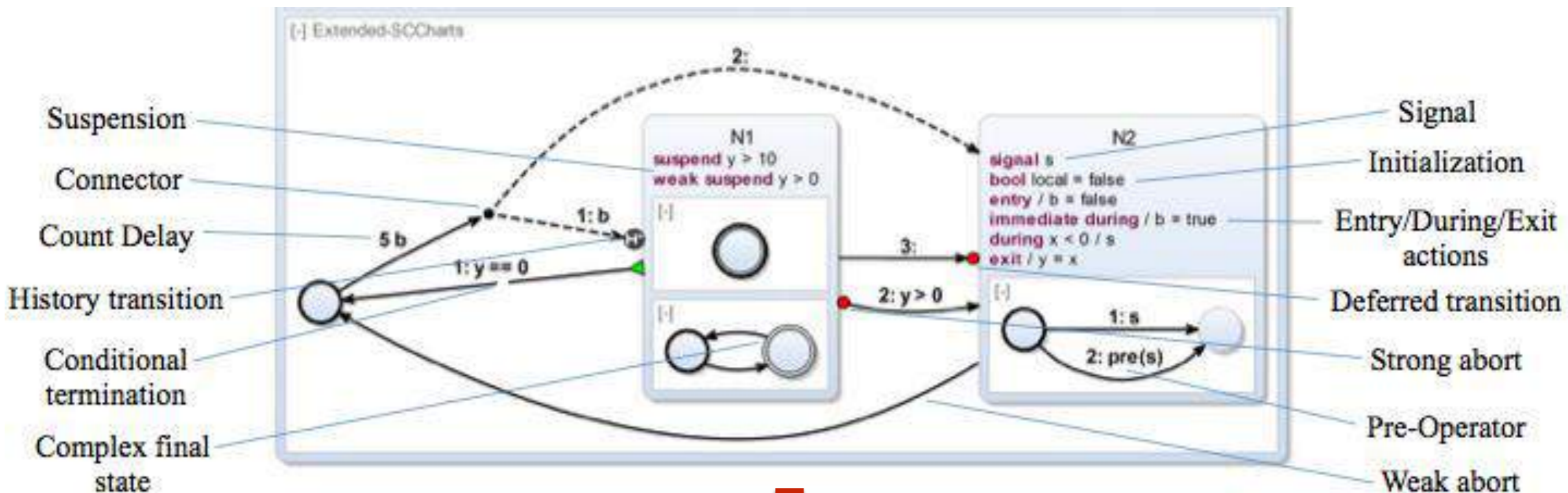
	Thread	Conditional	Assignment	Concurrency	Delay
SCL	$t$	if ( $C$ ) $S_1$ else $S_2$	$x = e$	fork $t_1$ par $t_2$ join	pause
SCG					

# Some Syntactic Sugar: Core SCCharts



	Region	Trigger	Effect	Superstate	State
<b>Normalized Core SCCharts</b>					

# More Syntactic Sugar: Extended SCCharts



# Compilation: Expand Signals

The image shows the KIELER IDE interface with two diagrams side-by-side, illustrating the expansion of signals during compilation.

**Left Diagram (Original ABRO):**

- Component: **ABRO**
- Inputs: **input signal A, B, R**; **output signal O**
- Sub-component: **ABthenO**
- Sub-sub-component: **WaitAandB**
- Internal states: **HandleA** (wA) and **HandleB** (wB)
- Transitions: **wA** to **dA** (labeled **A**), **wB** to **dB** (labeled **B**)
- Output: **done** state reached via transition **/ O**
- Feedback: **R** signal loops back to the input of **ABthenO**

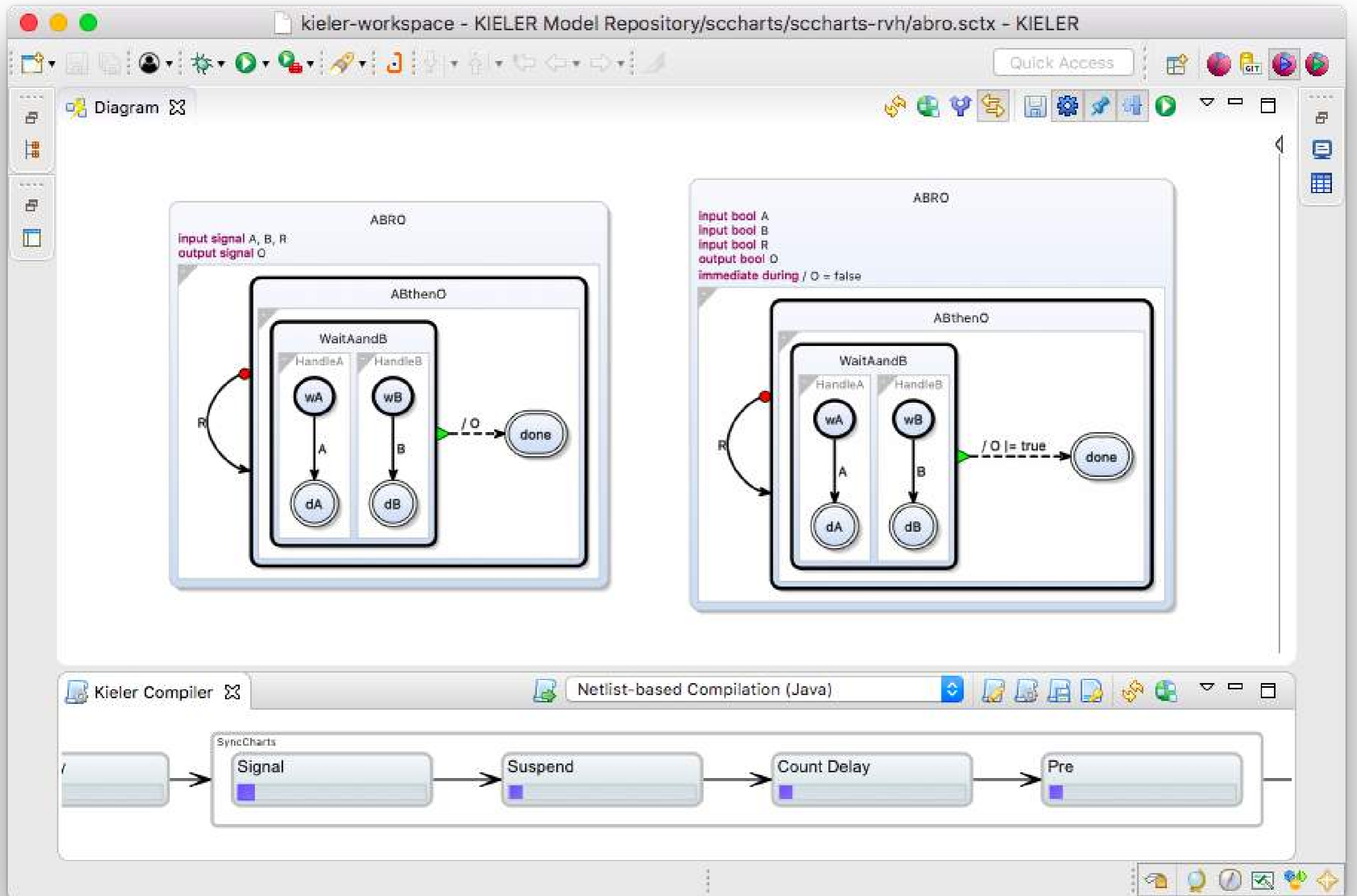
**Right Diagram (Expanded ABRO):**

- Component: **ABRO**
- Inputs: **input bool A**, **input bool B**, **input bool R**; **output bool O**
- Property: **immediate during / O = false**
- Sub-component: **ABthenO**
- Sub-sub-component: **WaitAandB**
- Internal states: **HandleA** (wA) and **HandleB** (wB)
- Transitions: **wA** to **dA** (labeled **A**), **wB** to **dB** (labeled **B**)
- Output: **done** state reached via transition **/ O |= true**
- Feedback: **R** signal loops back to the input of **ABthenO**

**Bottom Panel (Kieler Compiler):**

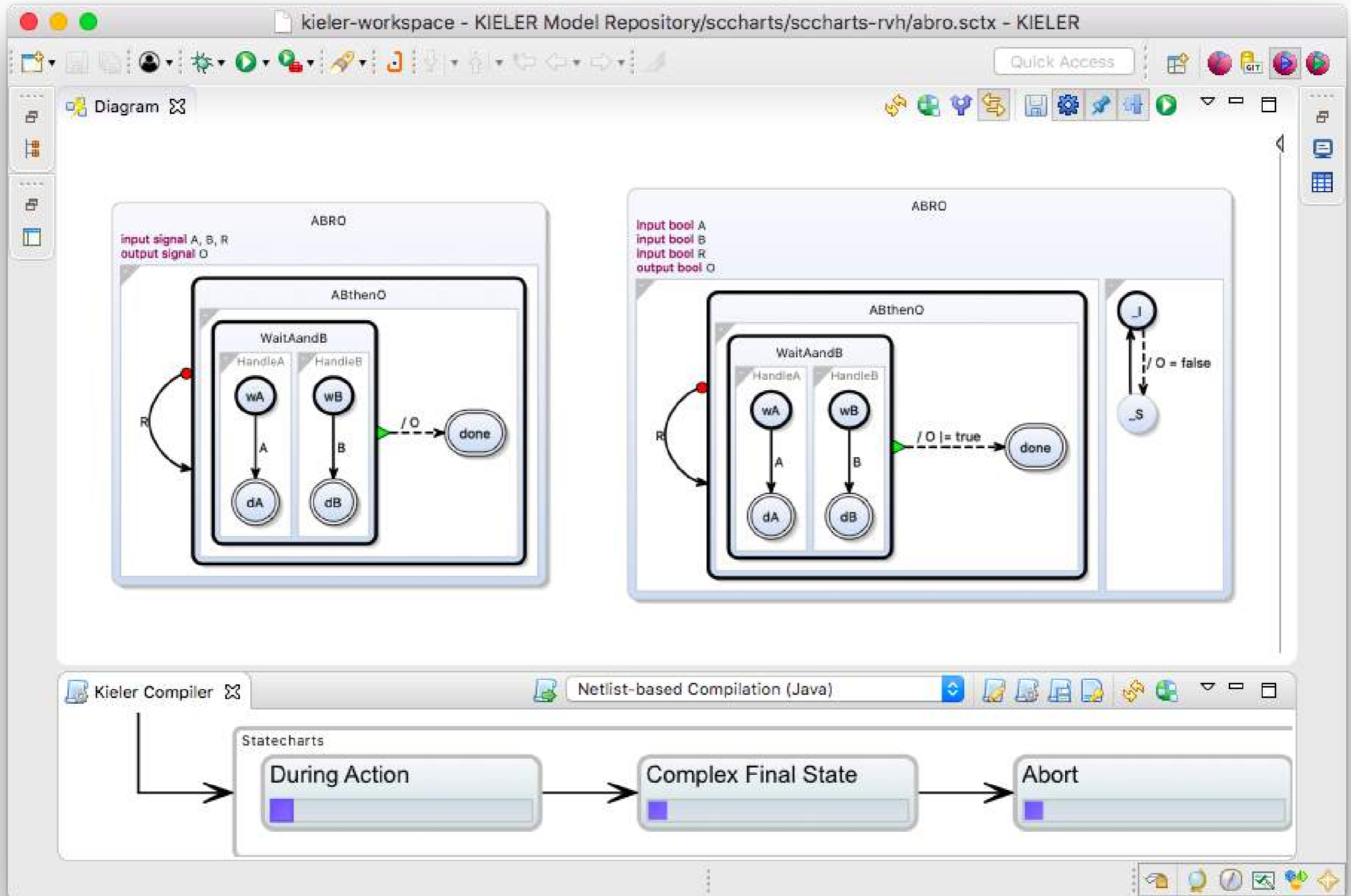
- Tool: **Netlist-based Compilation (Java)**
- Flowchart showing the compilation process: **Reference V2** → **Use V2** → **Goal** → **Followed By** → **Signal** → **Element** → **Count Delay** → **Pre**
- Below the flowchart, a sequence of elements is shown, including a **BOO** element.

# Compilation: Expand Signals

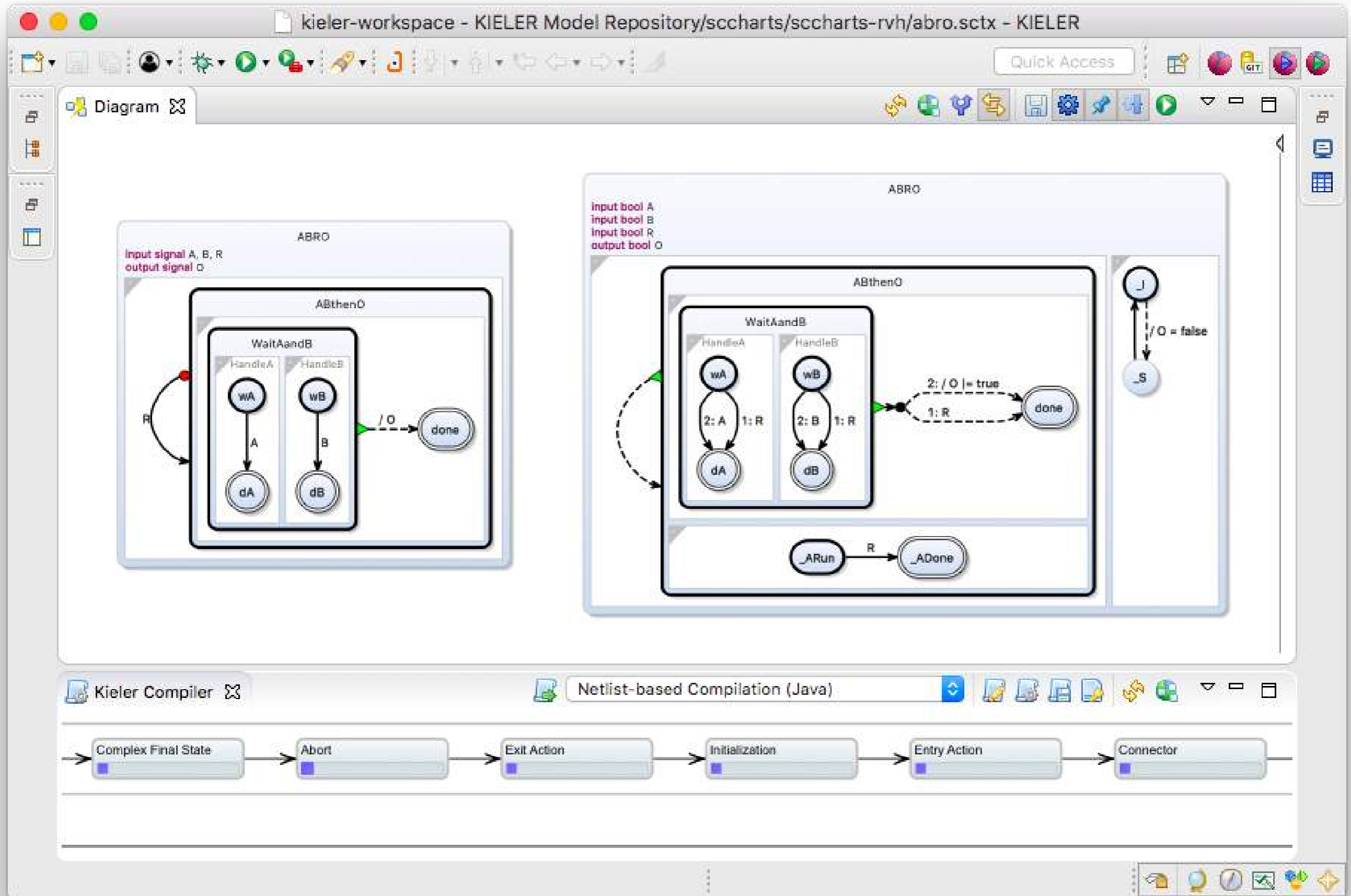




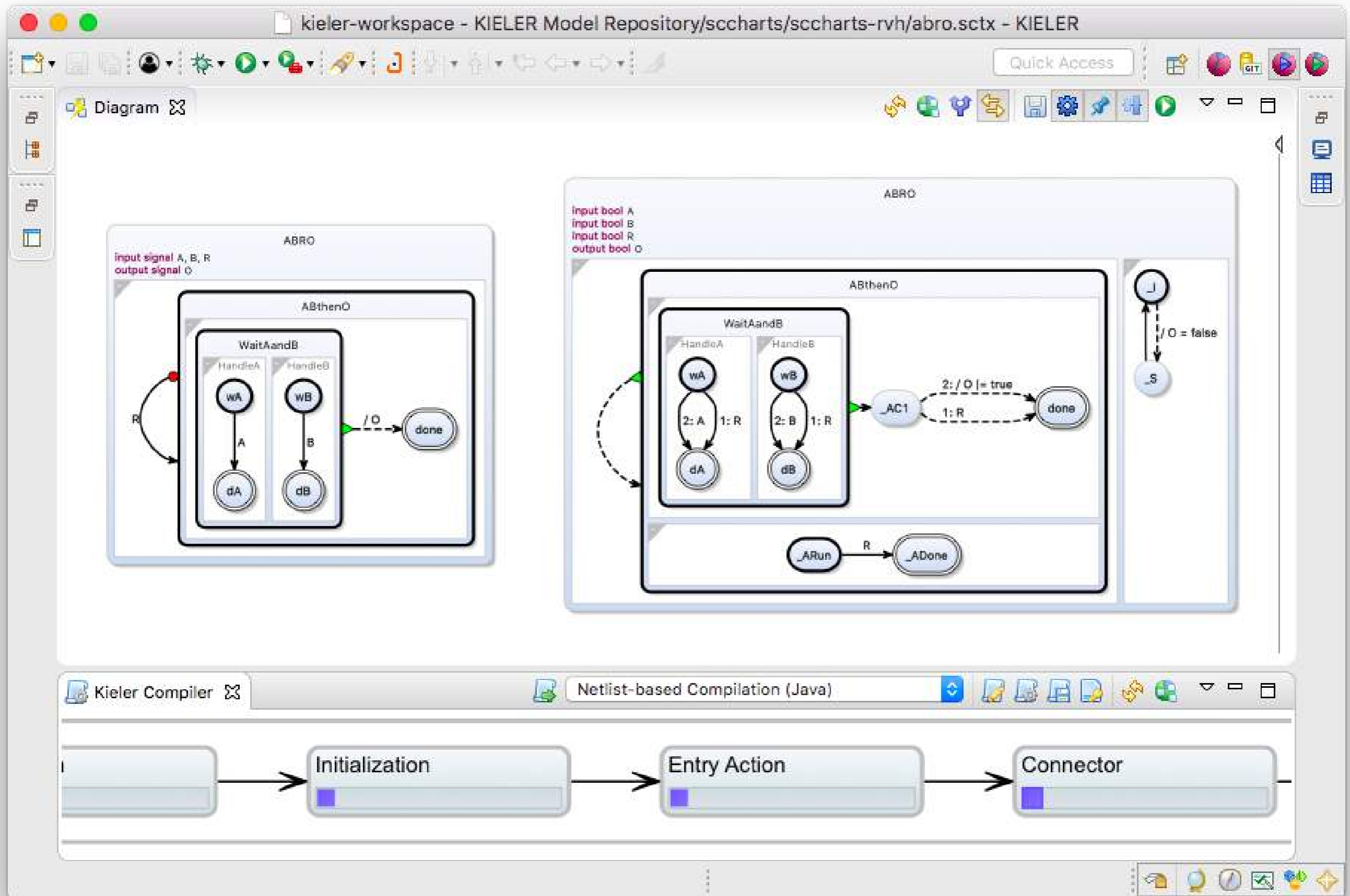
# Expand During Action



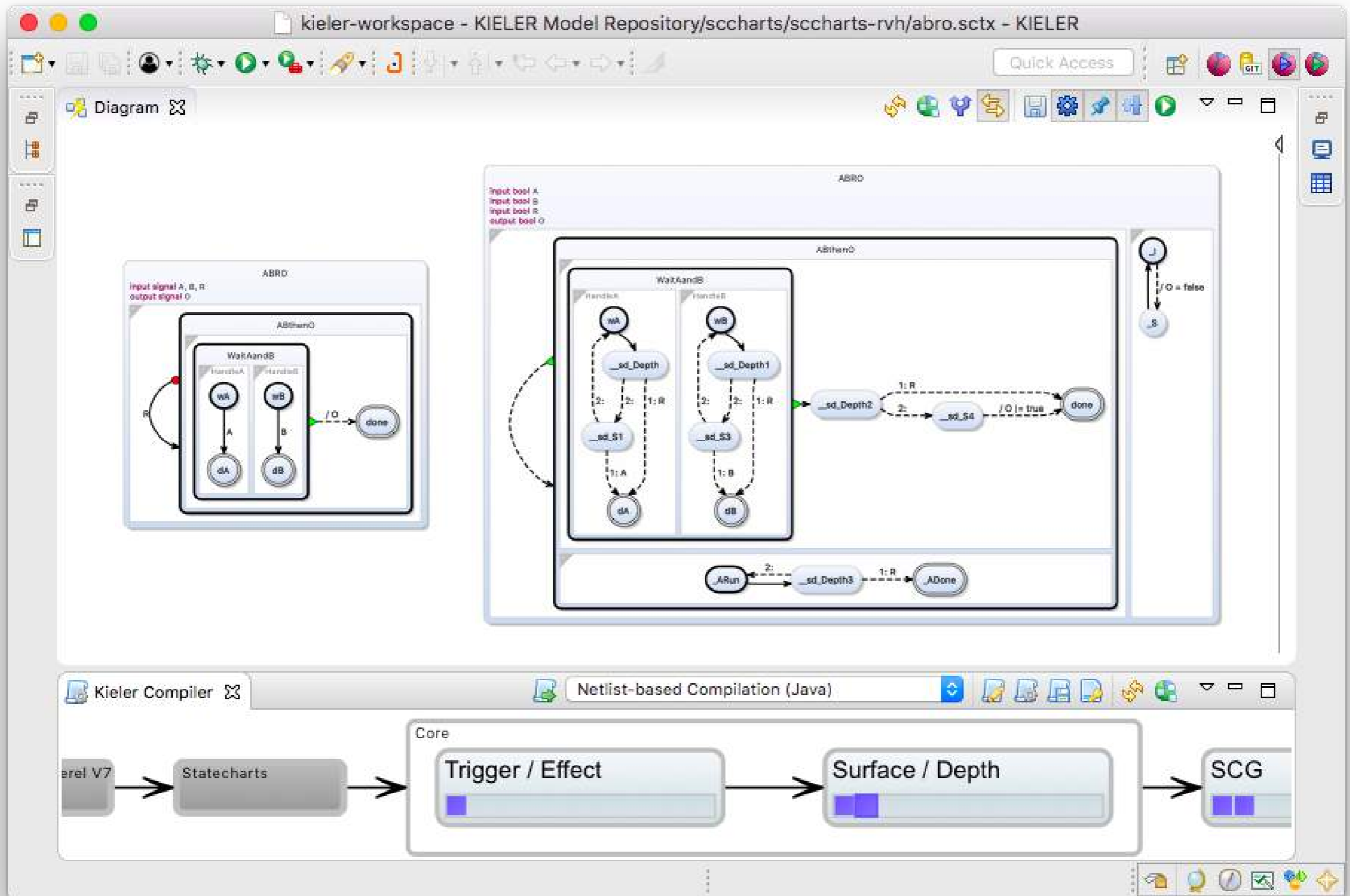
# Expand Abort



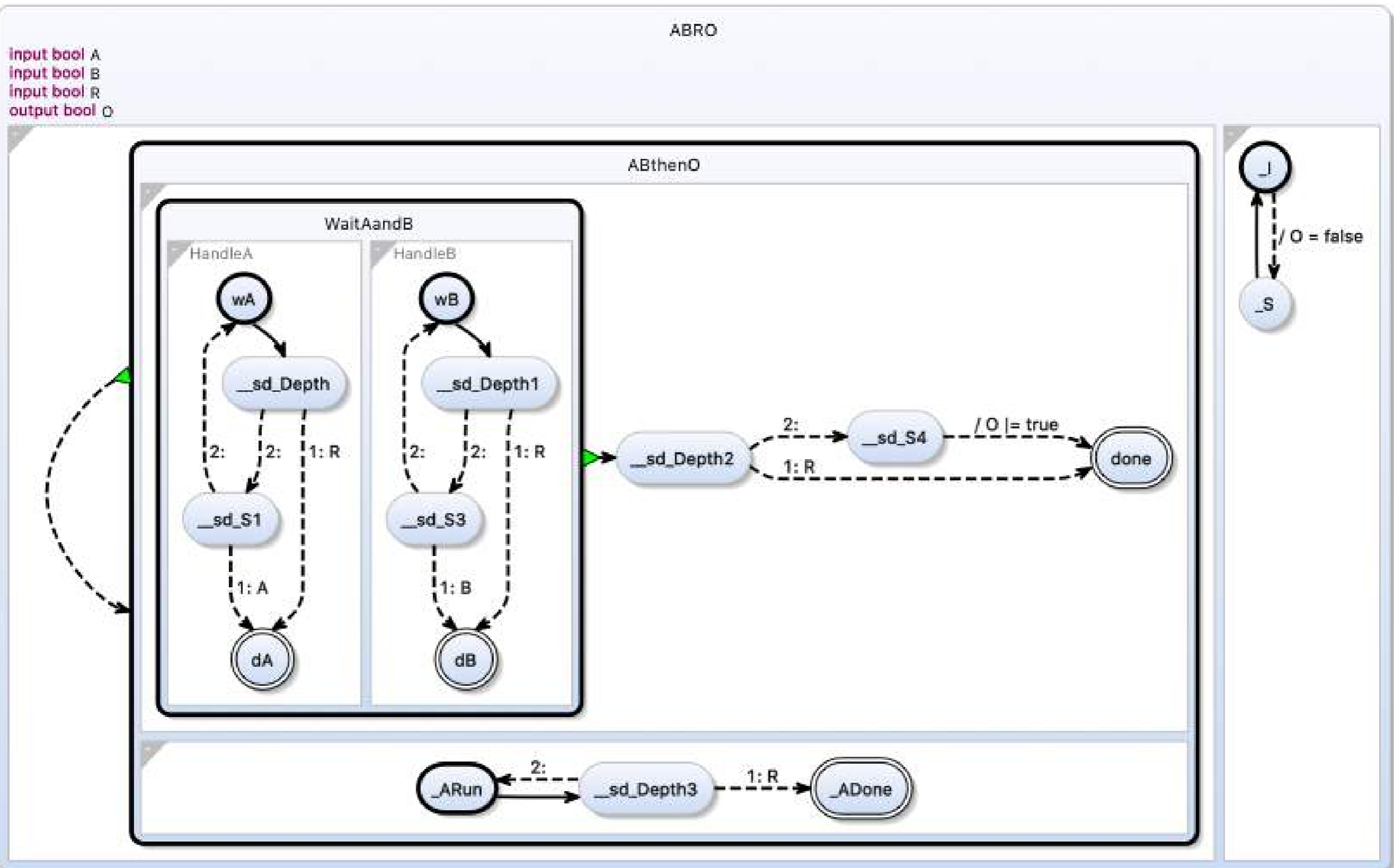
# Core SCChart



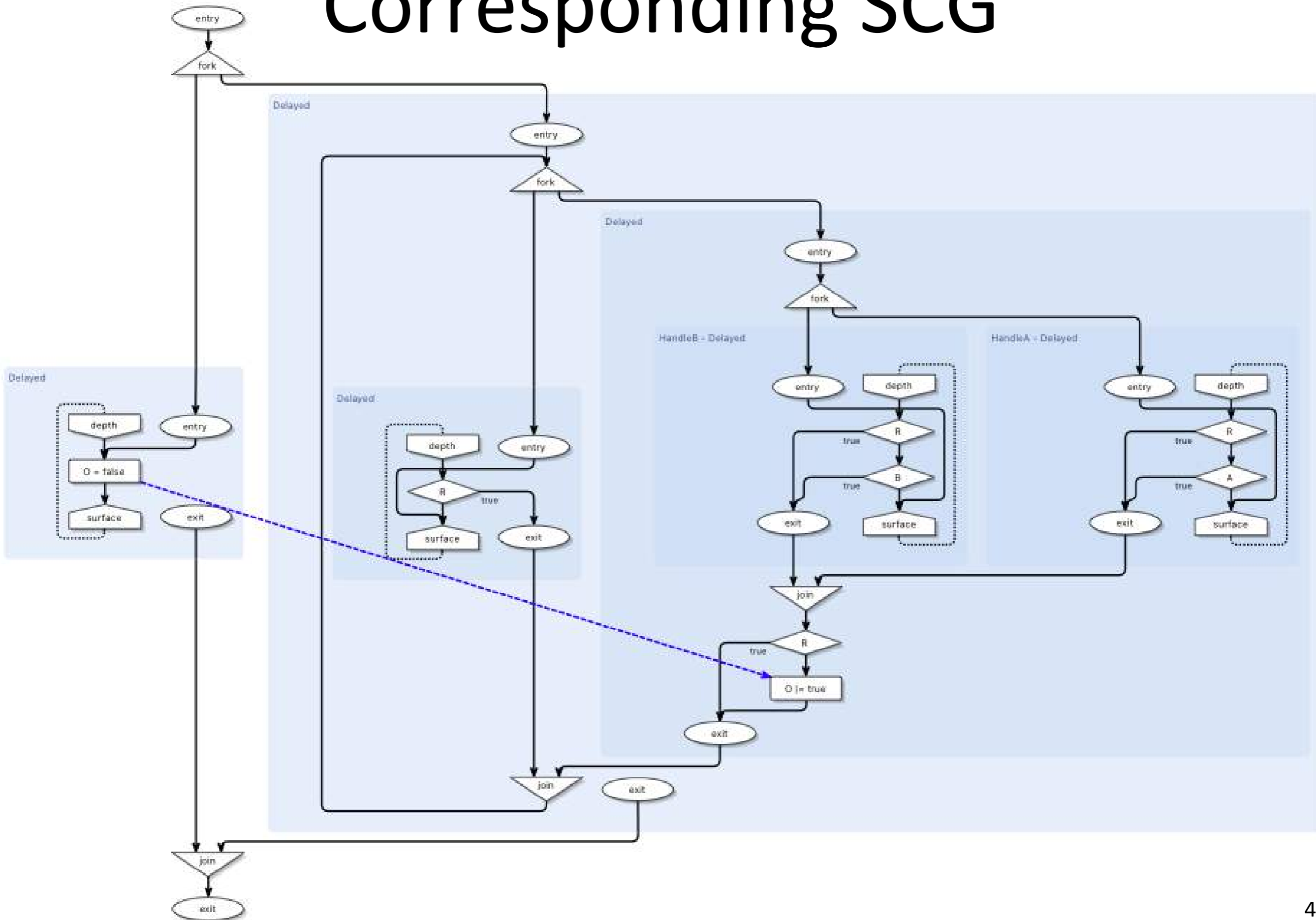
# Normalized Core SCChart



# Normalized Core SCChart



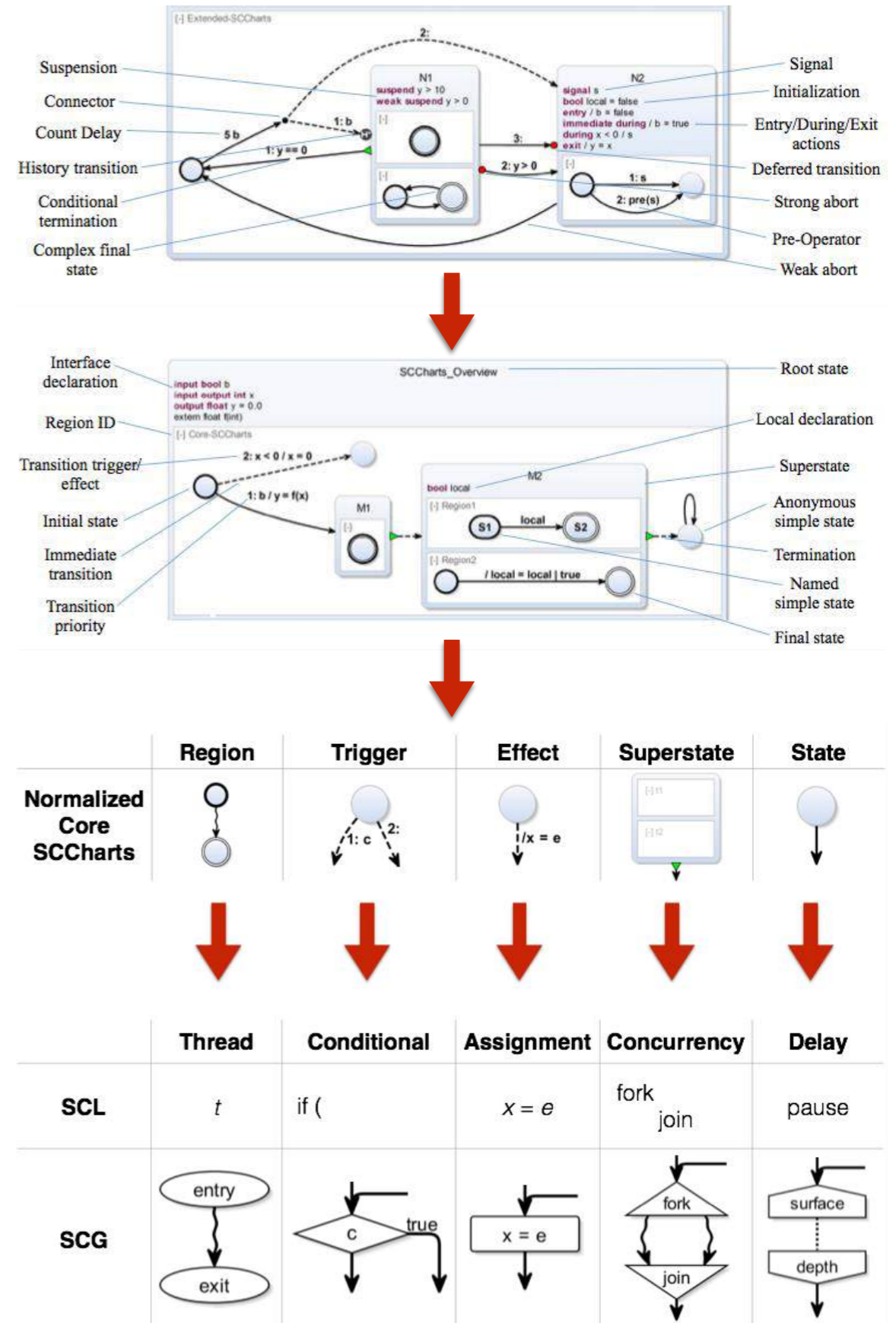
# Corresponding SCG



# Taking Stock

## SCCharts defined by M2M Transformations

- Extended SCCharts
- Core SCCharts
- Normalized Core SCCharts
- SCL/SCG



# SCCharts – Pragmatics



# Key to Pragmatics: MVC

- A **model** represents knowledge
- A **view** is a (visual) representation of its model.  
It would ordinarily highlight certain attributes of the model and suppress others.  
It is thus acting as a **presentation filter**.
- A **controller** is the link between a user and the system.  
It provides the user with input by **arranging for relevant views to present themselves in appropriate places on the screen**.

[Trygve Reenskaug,  
*Models – Views – Controllers*,  
Xerox PARC technical note, 1979]

# Textual Modeling

```
scchart ABRO {  
  input signal A, B, R  
  output signal O
```

```
  initial state ABthenO {
```

```
    initial state WaitAandB {  
      region "HandleA" {  
        initial state wA  
        if A go to dA
```

```
      }  
      final state dA
```

```
    }  
    region "HandleB" {  
      initial state wB  
      if B go to dB
```

```
    }  
    final state dB  
  }
```

```
  join to done do O
```

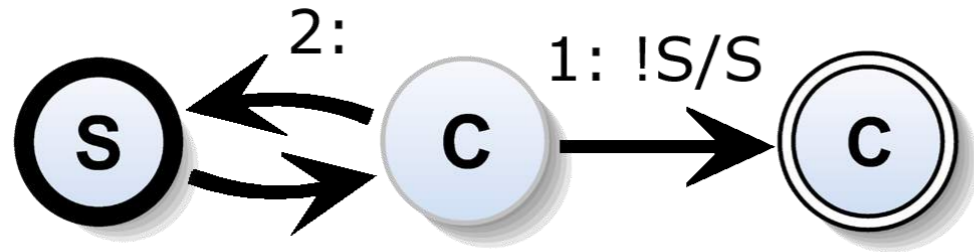
```
  final state done  
}  
if R abort to ABthenO  
}
```

SCChart **model** specified in .sctx

- Efficient editing
- Facilitates model comparison
- Easy revision control

Graphical **view** automatically synthesized with KIELER (**controller**)

- Customizable view
- Saves developer time



# SCCharts

<http://www.sccharts.com/>



# KIELER

The Key to Efficient Modeling

<http://www.rtsys.informatik.uni-kiel.de/en/research/kieler>

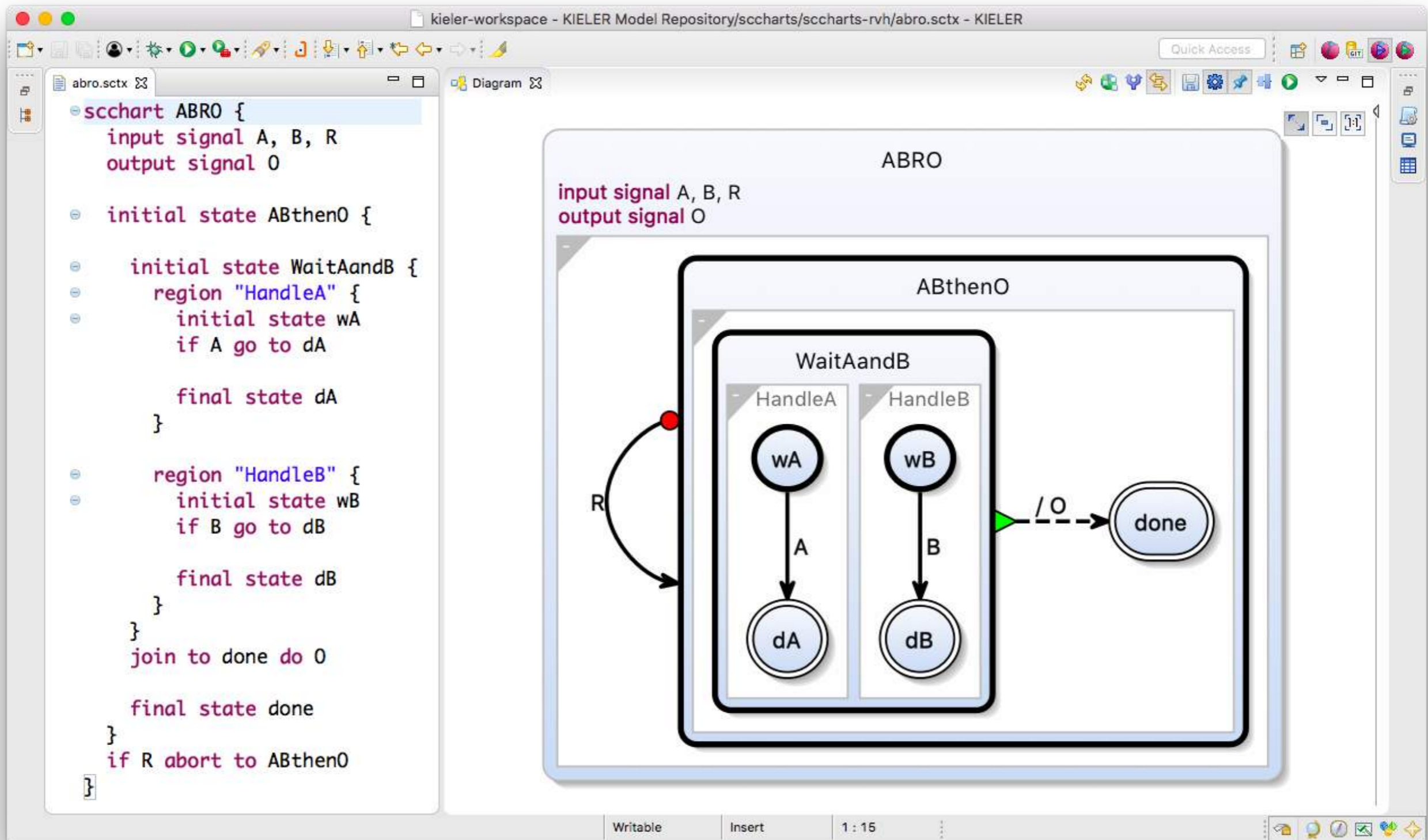


# Eclipse Layout Kernel

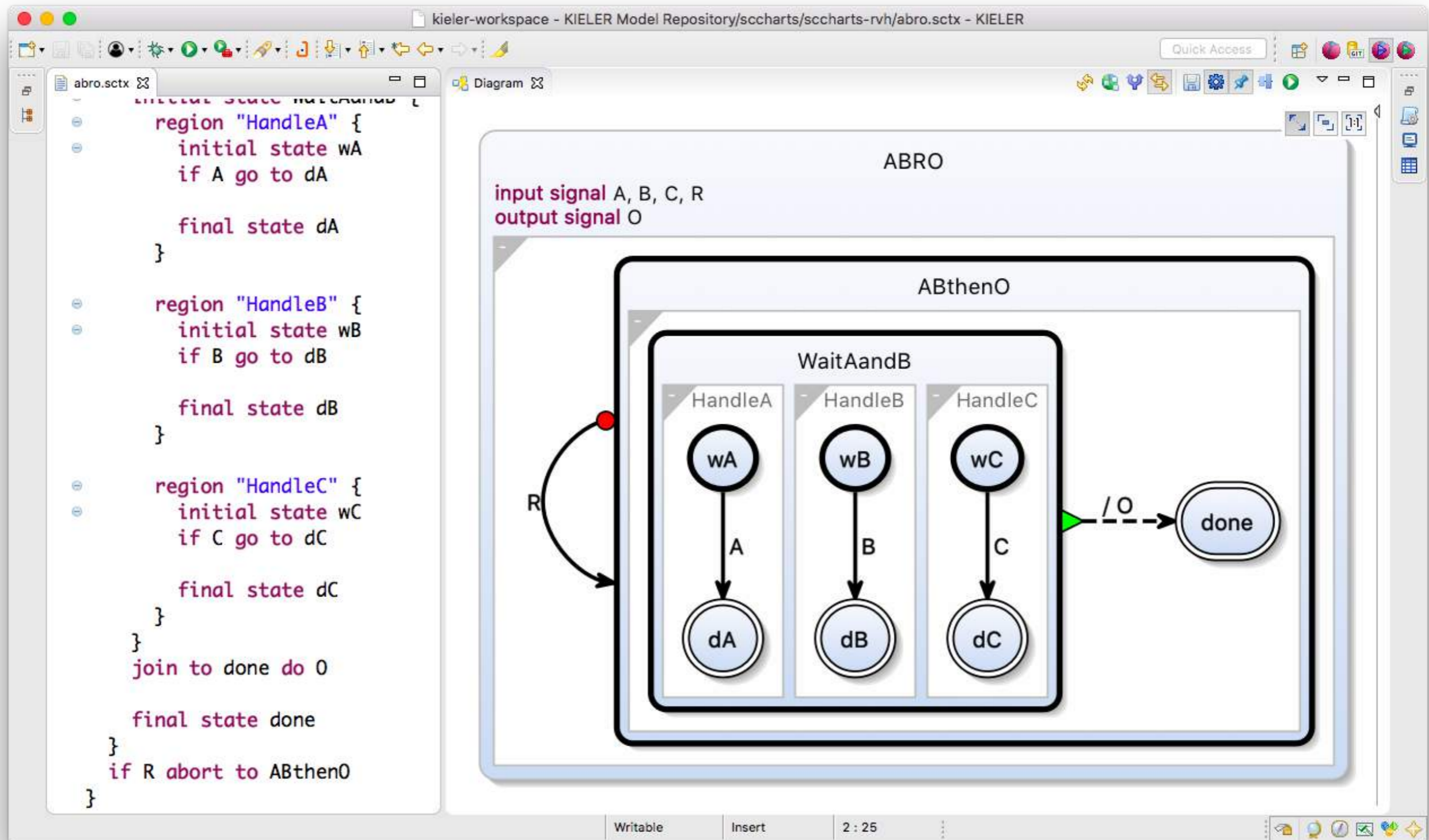
<https://www.eclipse.org/elk/>

All open source, under EPL license

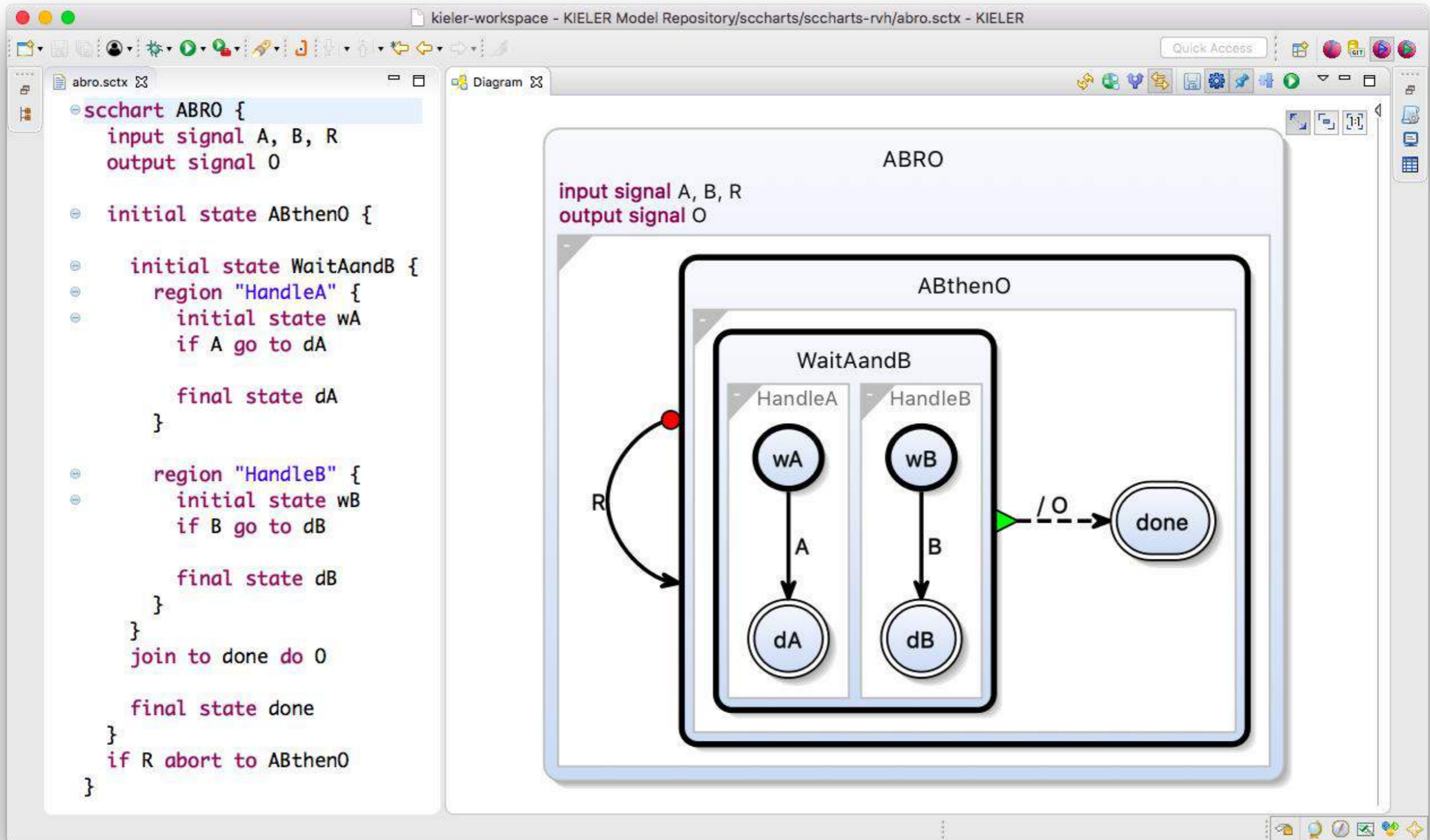
# View Synthesis



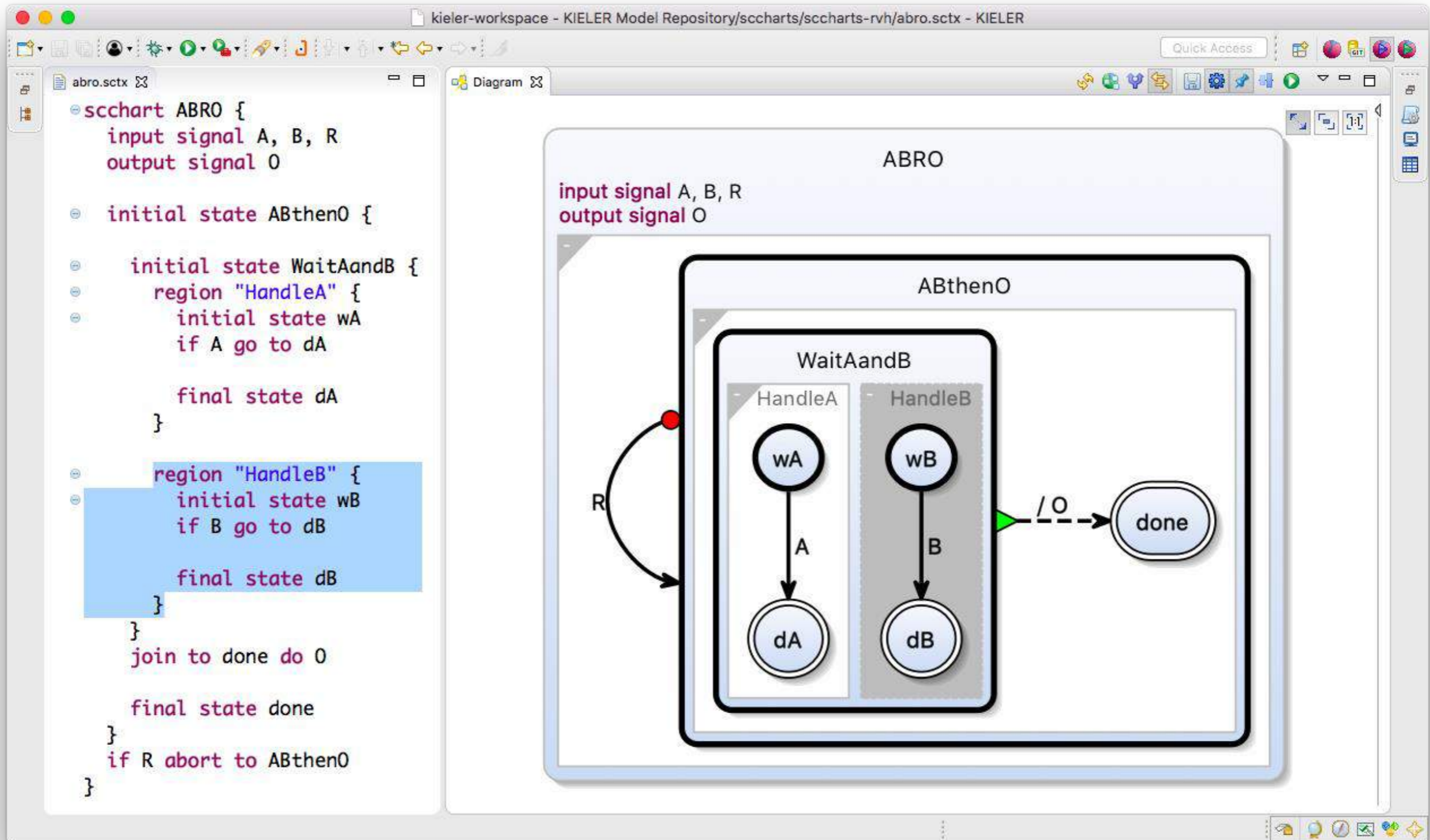
# Adding Region "HandleC"



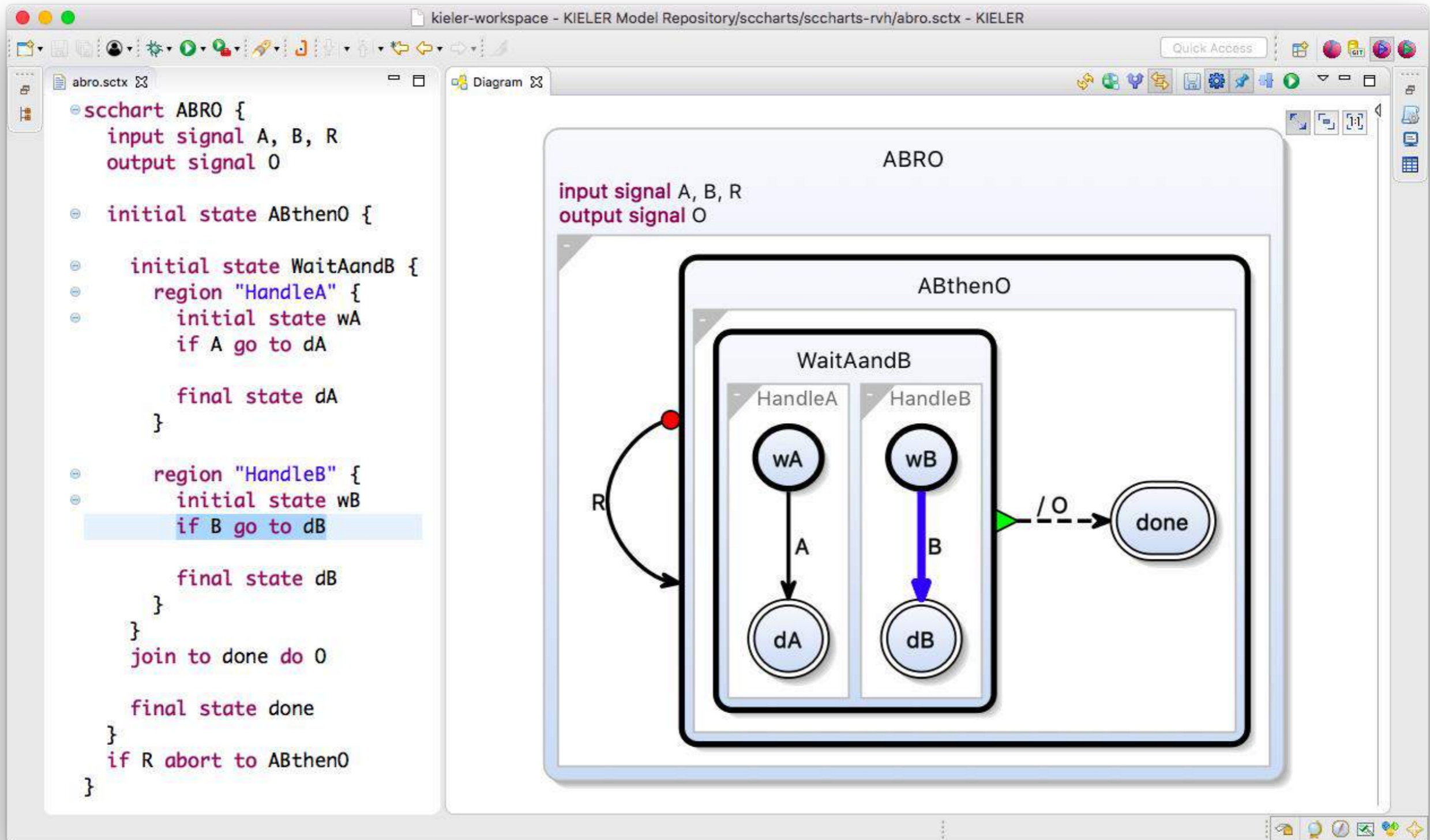
# Back to Original ABRO



# Graphics-to-Text Navigation: Select Region

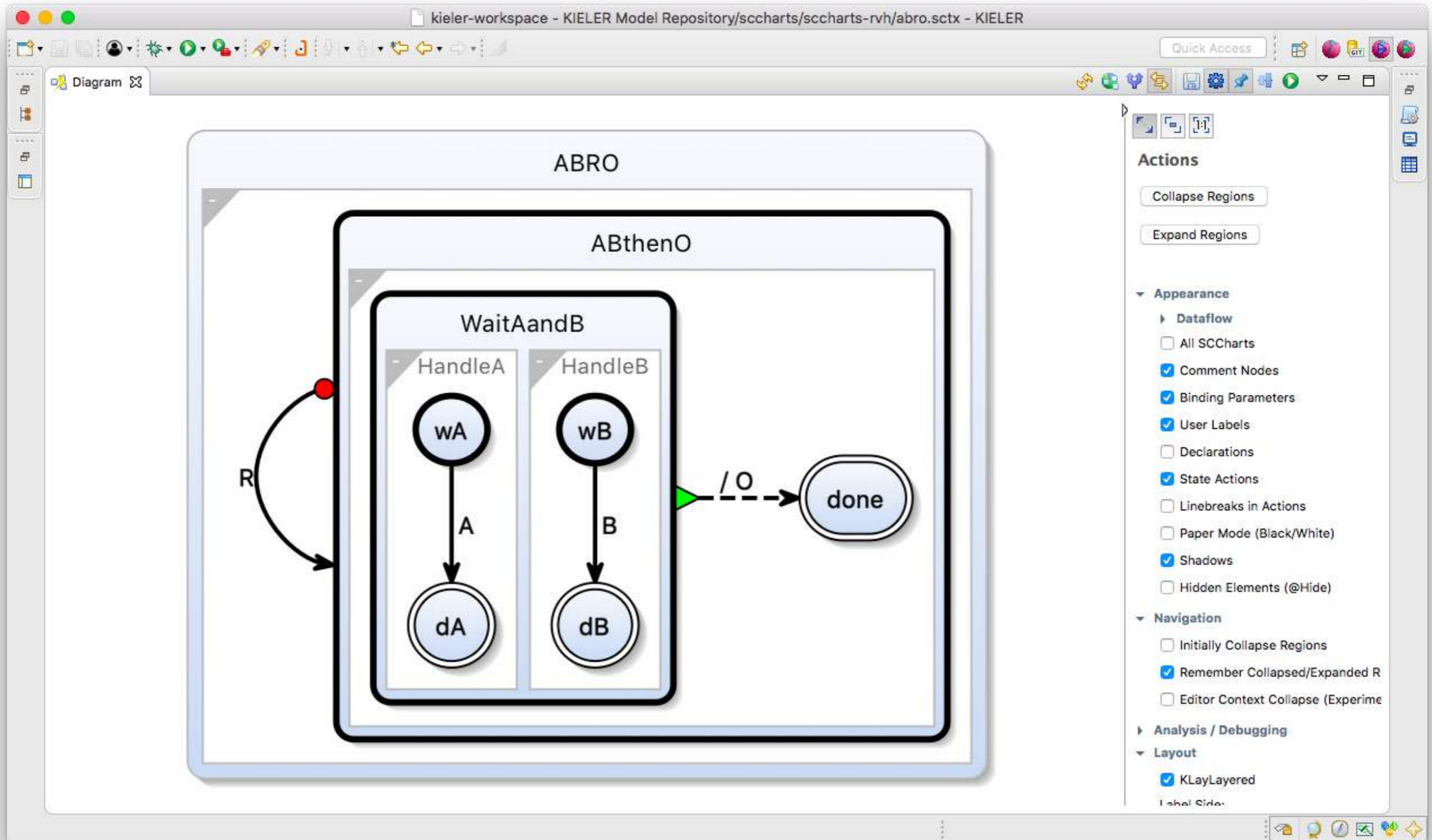


# Graphics-to-Text Navigation: Select Transition

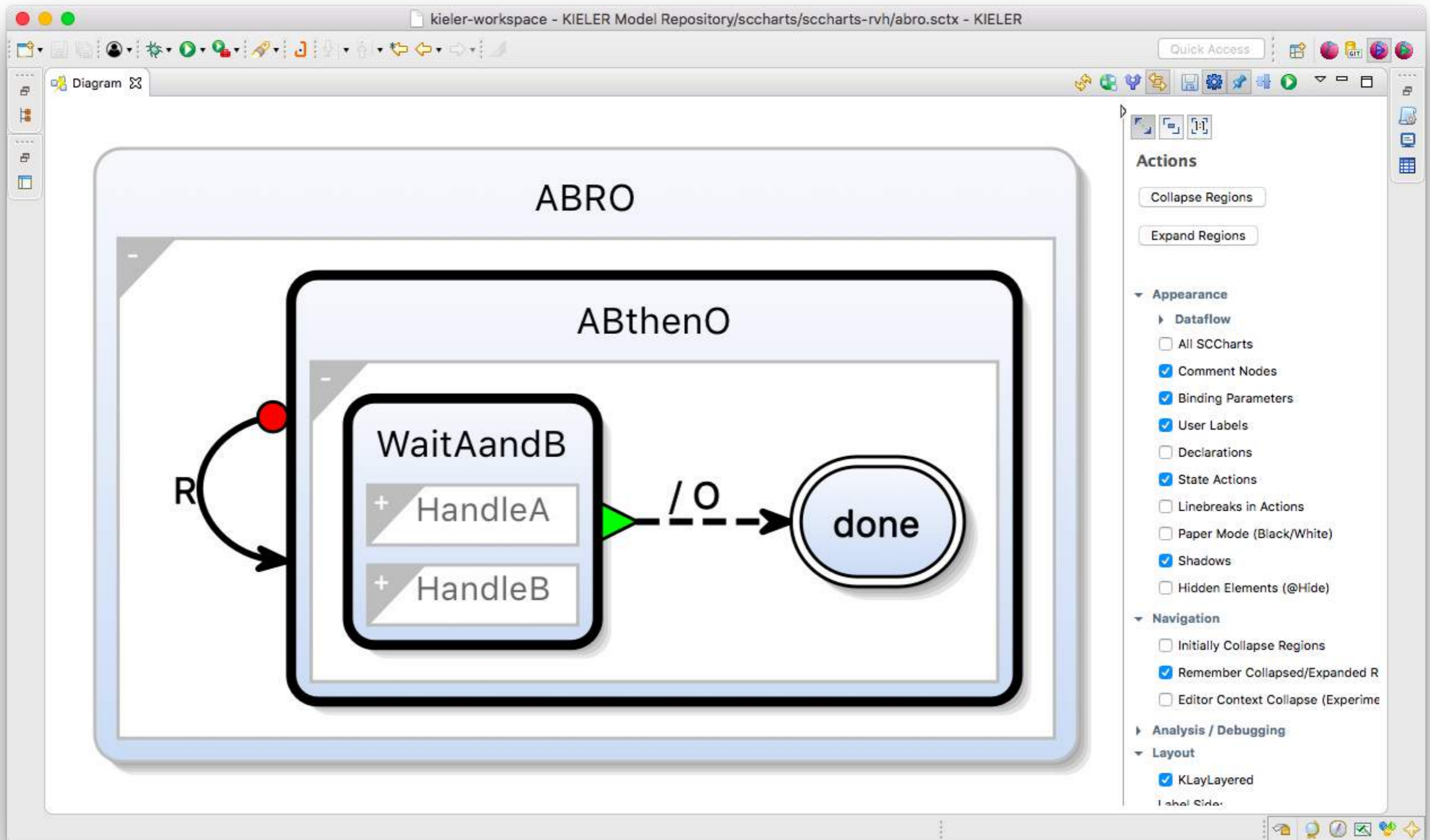




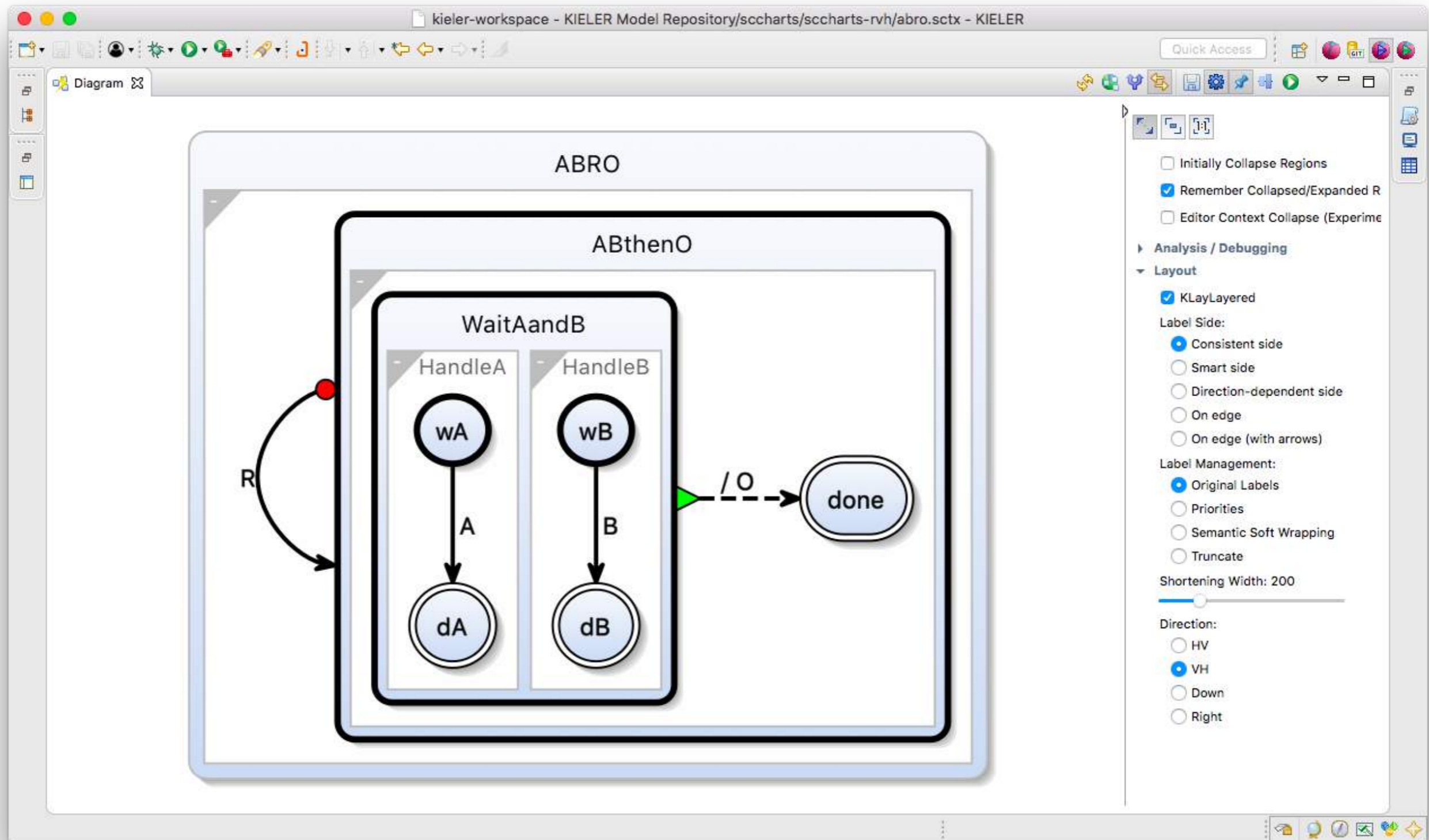
# View Filtering: Hide Declarations



# View Filtering: Collapse Regions



# Direction VH (Vertical-Horizontal)



# Change Layout: Direction HV

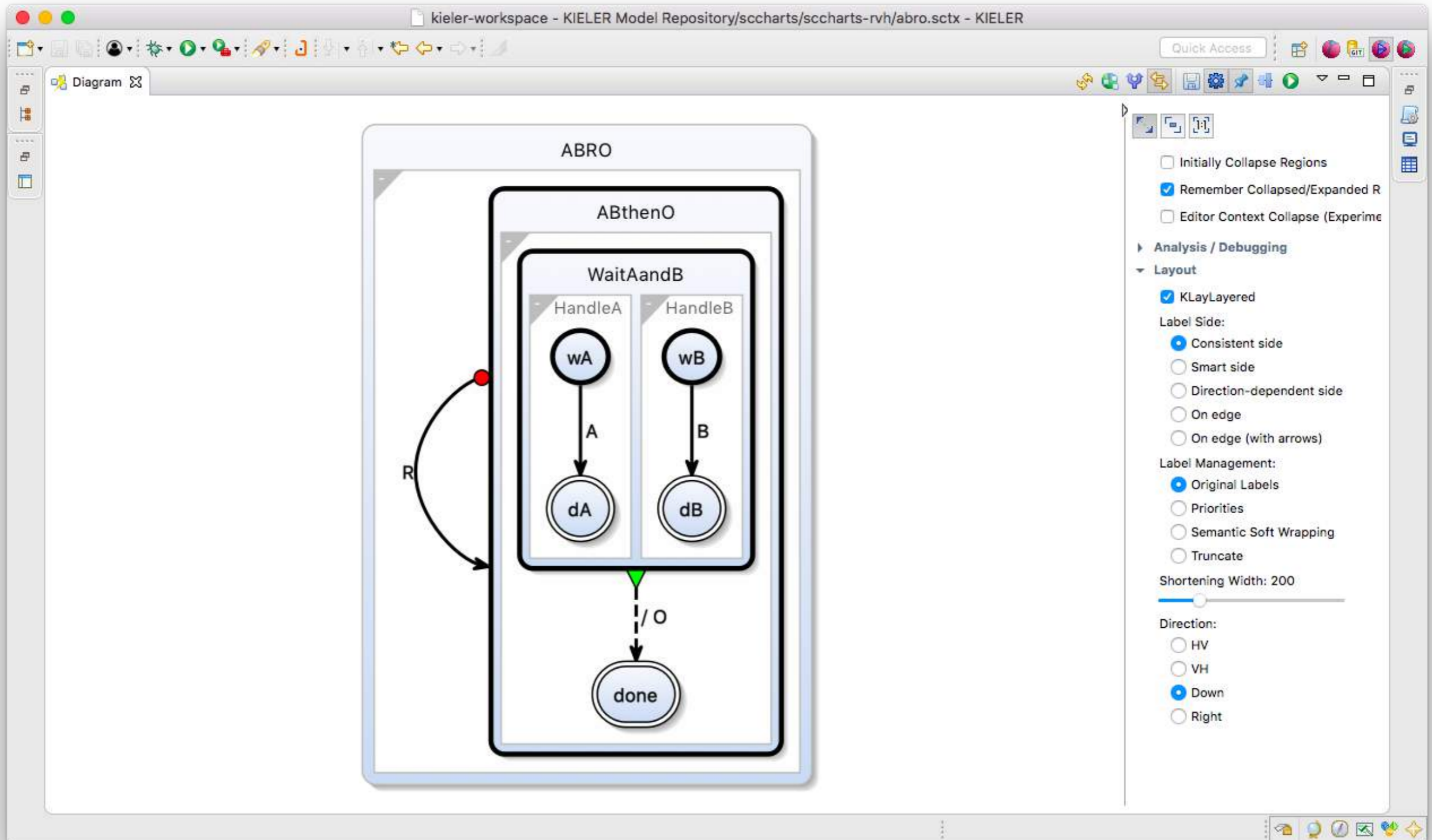
The screenshot displays the KIELER workspace with a state machine diagram for 'ABRO' and its configuration panel. The diagram is structured as follows:

- ABRO** (outermost region):
  - Contains a self-loop transition labeled **R** on a state.
  - Contains the **ABthenO** region.
- ABthenO** (middle region):
  - Contains the **WaitAandB** region.
  - Contains a transition labeled **/o** leading to the **done** state.
- WaitAandB** (innermost region):
  - HandleA** sub-region: contains states **wA** and **dA** with a transition labeled **A**.
  - HandleB** sub-region: contains states **wB** and **dB** with a transition labeled **B**.

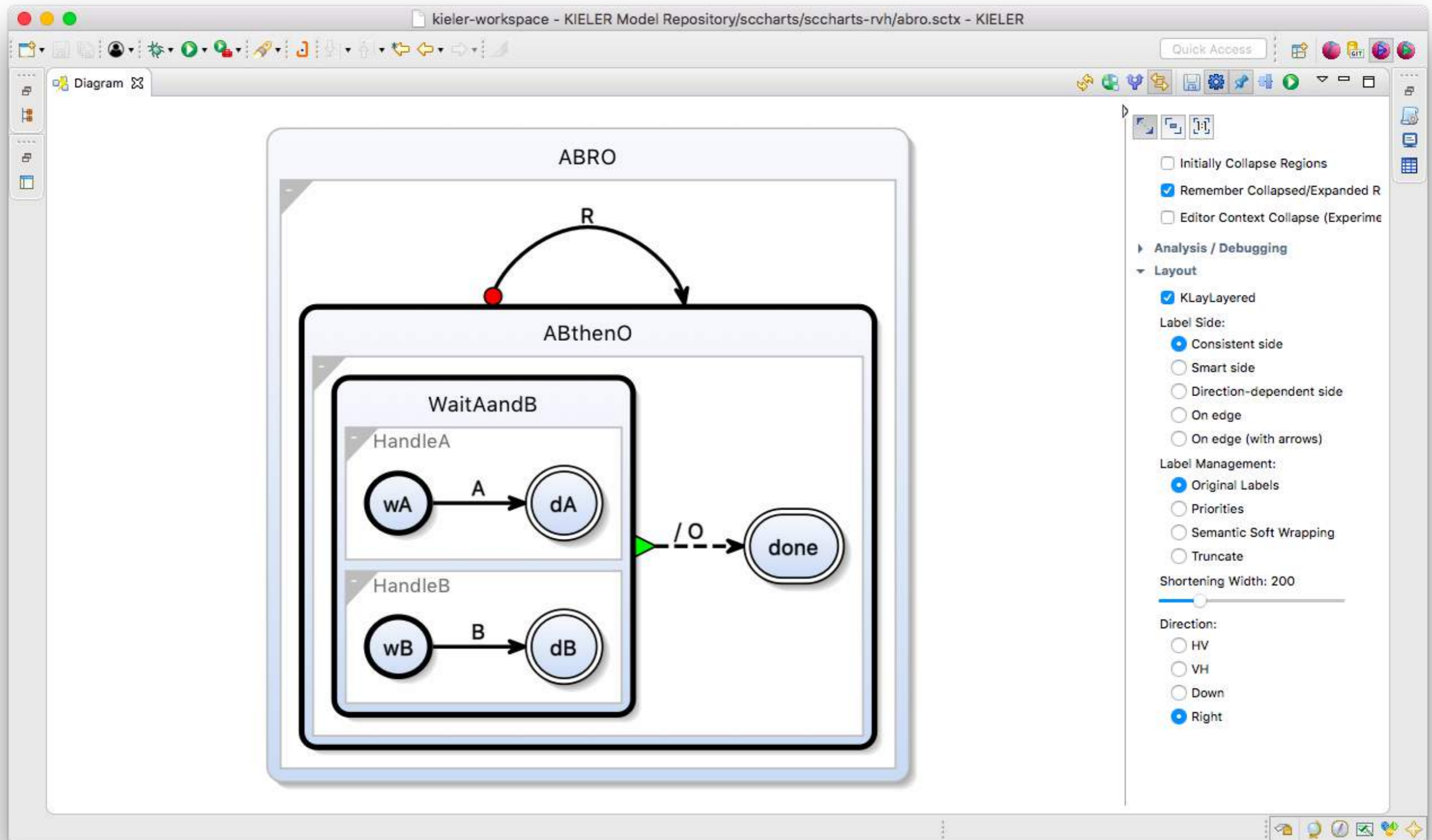
The configuration panel on the right shows the following settings:

- Initially Collapse Regions
- Remember Collapsed/Expanded R
- Editor Context Collapse (Experime)
- Analysis / Debugging**
- Layout**
  - KLayoutLayered
  - Label Side:
    - Consistent side
    - Smart side
    - Direction-dependent side
    - On edge
    - On edge (with arrows)
  - Label Management:
    - Original Labels
    - Priorities
    - Semantic Soft Wrapping
    - Truncate
  - Shortening Width: 200
  - Direction:
    - HV
    - VH
    - Down
    - Right

# Change Layout: Direction Down



# Change Layout: Direction Right



# Simulation – Initialization

The screenshot displays the KIELER workspace interface. The main diagram, titled "ABRO", is a Petri net with the following structure:

- ABRO** (outermost container):
  - Input signal  $A, B, R$  and output signal  $O$ .
  - ABthenO** (middle container):
    - WaitAandB** (inner container):
      - HandleA**: Contains transition  $wA$  and place  $dA$ . Transition  $wA$  is connected to place  $dA$  via signal  $A$ .
      - HandleB**: Contains transition  $wB$  and place  $dB$ . Transition  $wB$  is connected to place  $dB$  via signal  $B$ .
    - Transition  $/O$  is connected to place  $done$ .

The simulation visualization on the right shows the state of the system over time. It includes a "Data View" with four signal traces:  $A$ ,  $B$ ,  $R$ , and  $O$ . Each trace has a vertical axis with values 0 and 1. The  $R$  signal starts at 1, then drops to 0. The  $A$  and  $B$  signals start at 0. The  $O$  signal starts at 0.

Variable	Current Value	User Value

Signal	Value
A	0
B	0
R	1
O	0

# Simulation – Tick 1

kieler-workspace - KIELER

Diagram Simulation Visualization

ABRO

input signal A, B, R  
output signal O

ABthenO

WaitAandB

HandleA HandleB

wA wB

A (false) B (false)

dA dB

done

R (false)

/O

Quick Access

Kieler C Consol Data Po

Tick #1 500

Variable	Current Value	User Value
-----Sim_ABRO-----		
A	false	
B	false	
R	false	
O	false	

Data View

A

B

R

O



# Simulation – Tick 1

The screenshot displays the KIELER workspace with a simulation of an ABRO component. The diagram shows the following structure:

- ABRO** (outermost component):
  - Input signals: A, B, R
  - Output signal: O
  - Contains **ABthenO** component.
- ABthenO** (middle component):
  - Contains **WaitAandB** component.
  - Contains a **done** state.
- WaitAandB** (innermost component):
  - Contains **HandleA** and **HandleB** components.
  - HandleA leads to state **dA**.
  - HandleB leads to state **dB**.

The simulation console shows the following variable values at Tick #1:

Variable	Current Value	User Value	His
-----Sim_ABRO...			
*A	false	true	
B	false		
R	false		
O	false		

The data view shows plots for variables A, B, R, and O, all currently at 0.

# Simulation – Tick 2

The screenshot displays the KIEMER simulation environment. The main window shows a state machine diagram for a component named 'ABRO'. The diagram is organized into nested boxes: 'ABRO' (outermost), 'ABthenO' (middle), and 'WaitAandB' (innermost). The 'WaitAandB' box contains two parallel state machines: 'HandleA' and 'HandleB'. 'HandleA' has a state 'wA' (blue circle) that transitions to 'dA' (red double circle) upon receiving input 'A (true)'. 'HandleB' has a state 'wB' (red circle) that transitions to 'dB' (blue double circle) upon receiving input 'B (false)'. A red dot on the left indicates the current state, with an arrow labeled 'R (false)' pointing to it. A dashed arrow labeled 'O' points from the 'done' state (a double circle) to the right. The simulation is currently at 'Tick #2' with a value of 500.

input signal A, B, R  
output signal O

ABRO

ABthenO

WaitAandB

HandleA

HandleB

wA

wB

dA

dB

done

R (false)

A (true)

B (false)

O

Tick #2

500

Variable	Current Value	User Value	His
-----Sim_ABRO...			
A	true		fal
B	false		fal
R	false		fal
O	false		fal

Data View

A

B

R

O

# Simulation – Tick 3

The screenshot displays the KIELER workspace with a simulation of an ABRO component. The diagram shows the internal state of the component at Tick #3. The component is labeled "ABRO" and has input signals A, B, R and output signal O. The internal state is divided into three main sections: "WaitAandB", "HandleA", and "HandleB".

- WaitAandB:** Contains two waiters, wA and wB. wA is currently active (highlighted in red), and wB is inactive (highlighted in blue).
- HandleA:** Contains a handler dA. The transition from wA to dA is triggered by the event "A (true)". dA is currently active (highlighted in red).
- HandleB:** Contains a handler dB. The transition from wB to dB is triggered by the event "B (false)". dB is currently inactive (highlighted in blue).

The output signal O is currently false. A feedback loop labeled "R (false)" connects the output back to the input R. The simulation visualization on the right shows the current values of the variables A, B, R, and O, along with their history over time.

Variable	Current Value	User Value	His
-----Sim_ABRO...			
A	true		fal
B	false		fal
R	false		fal
O	false		fal

The Data View section shows four signal traces: A, B, R, and O. Trace A shows a step function that transitions from 0 to 1 at the start of the simulation. Trace B shows a step function that transitions from 1 to 0 at the start of the simulation. Trace R shows a step function that transitions from 1 to 0 at the start of the simulation. Trace O shows a step function that transitions from 0 to 1 at the start of the simulation.

# Simulation – Tick 3

kieler-workspace - KIELER

Diagram Simulation Visualization

ABRO

input signal A, B, R  
output signal O

ABthenO

WaitAandB

HandleA HandleB

wA wB

A (true) B (false)

dA dB

done

R (false)

/O

Tick #3

Variable	Current Value	User Value	His
-----Sim_ABRO...			
A	true		fal
*B	false	true	fal
R	false		fal
O	false		fal

Data View

A

B

R

O

# Simulation – Tick 4

The screenshot displays the KIELER workspace with a simulation of an ABRO component. The diagram shows the following structure:

- ABRO** (outermost container):
  - Input signals: A, B, R
  - Output signal: O
  - ABthenO** (red border):
    - WaitAandB** (blue border):
      - HandleA**: Contains state **wA** (waiting) and **dA** (done). Transition: **wA** to **dA** on **A (true)**.
      - HandleB**: Contains state **wB** (waiting) and **dB** (done). Transition: **wB** to **dB** on **B (true)**.
    - Transition from **WaitAandB** to **done** on **/O**.
  - done** (red double circle): Final state.

The simulation is at **Tick #4**. The **Data View** shows the following signal values:

Variable	Current Value	User Value	His
-----Sim_ABRO...			
A	true		fal
B	true		fal
R	false		fal
O	true		fal

The **Console** shows the current state of variables:

Variable	Current Value	User Value	His
-----Sim_ABRO...			
A	true		fal
B	true		fal
R	false		fal
O	true		fal

The **Data View** shows four signal waveforms: **A**, **B**, **R**, and **O**. All signals are high (1) at Tick #4.

# Simulation – Tick 5

The screenshot displays the KIELER workspace with a simulation of an ABRO component. The diagram shows the following structure:

- ABRO** (outermost container)
  - Input signals: A, B, R
  - Output signal: O
  - ABthenO** (red border)
    - WaitAandB** (black border)
      - HandleA**: State **wA** transitions to **dA** on event **A (true)**.
      - HandleB**: State **wB** transitions to **dB** on event **B (true)**.
    - Output **O** is produced when **dA** or **dB** occurs.
    - Final state: **done** (red oval).

The simulation console shows the state at Tick #5:

Variable	Current Value	User Value	His
-----Sim_ABRO...			
A	true		fal
B	true		fal
R	false		fal
O	false		fal

The Data View shows waveforms for signals A, B, R, and O. A and B are high, R is low, and O has a single pulse.

# Simulation – Tick 5

The screenshot displays the KIELER simulation workspace. The main diagram, titled "ABRO", shows a state machine with the following components:

- Input signals:** A, B, R
- Output signal:** O
- ABthenO:** A red-bordered container containing:
  - WaitAandB:** A black-bordered container with two parallel paths:
    - HandleA:** Contains state **wA** (waiting) and state **dA** (done). A transition from **wA** to **dA** is labeled "A (true)".
    - HandleB:** Contains state **wB** (waiting) and state **dB** (done). A transition from **wB** to **dB** is labeled "B (true)".
  - done:** A red-bordered state reached from the **dA** and **dB** states via a transition labeled "O".
- External transition:** A curved arrow labeled "R (false)" points from the **done** state back to the **WaitAandB** container.

The simulation console on the right shows the state at Tick #5:

Variable	Current Value	User Value	His
-----Sim_ABRO...			
A	true		fal
B	true		fal
*R	false	true	fal
O	false		fal

The Data View on the right shows four signal traces:

- A:** A step function that transitions from 0 to 1 at the start of the simulation.
- B:** A step function that transitions from 0 to 1 shortly after A.
- R:** A signal that is 0 for most of the simulation but transitions to 1 at the end.
- O:** A signal that transitions from 0 to 1 when both A and B are 1, and returns to 0 when R becomes 1.

# Simulation – Tick 6

The screenshot displays the KIELER simulation workspace. The main diagram, titled "ABRO", shows a state machine with the following structure:

- ABRO** (outermost box):
  - Input signals: A, B, R
  - Output signal: O
  - Contains **ABthenO** (middle box):
    - Contains **WaitAandB** (innermost box):
      - HandleA**: State **wA** (red circle) transitions to **dA** (blue circle) on event **A (true)**.
      - HandleB**: State **wB** (red circle) transitions to **dB** (blue circle) on event **B (true)**.
    - Transitions from **WaitAandB** to **done** (blue circle) are triggered by **/O**.

A blue arrow labeled **R (true)** points to the initial state of the **WaitAandB** block.

On the right, the simulation console shows the state at **Tick #6** (500):

Variable	Current Value	User Value	His
-----Sim_ABRO...			
A	true		fal
B	true		fal
R	true		fal
O	false		fal

Below the console, the **Data View** shows four signal waveforms: **A**, **B**, **R**, and **O**. Waveforms for A, B, and R show a step function from 0 to 1. Waveform for O shows a single pulse from 0 to 1.



# SCCharts – Classroom-Tested

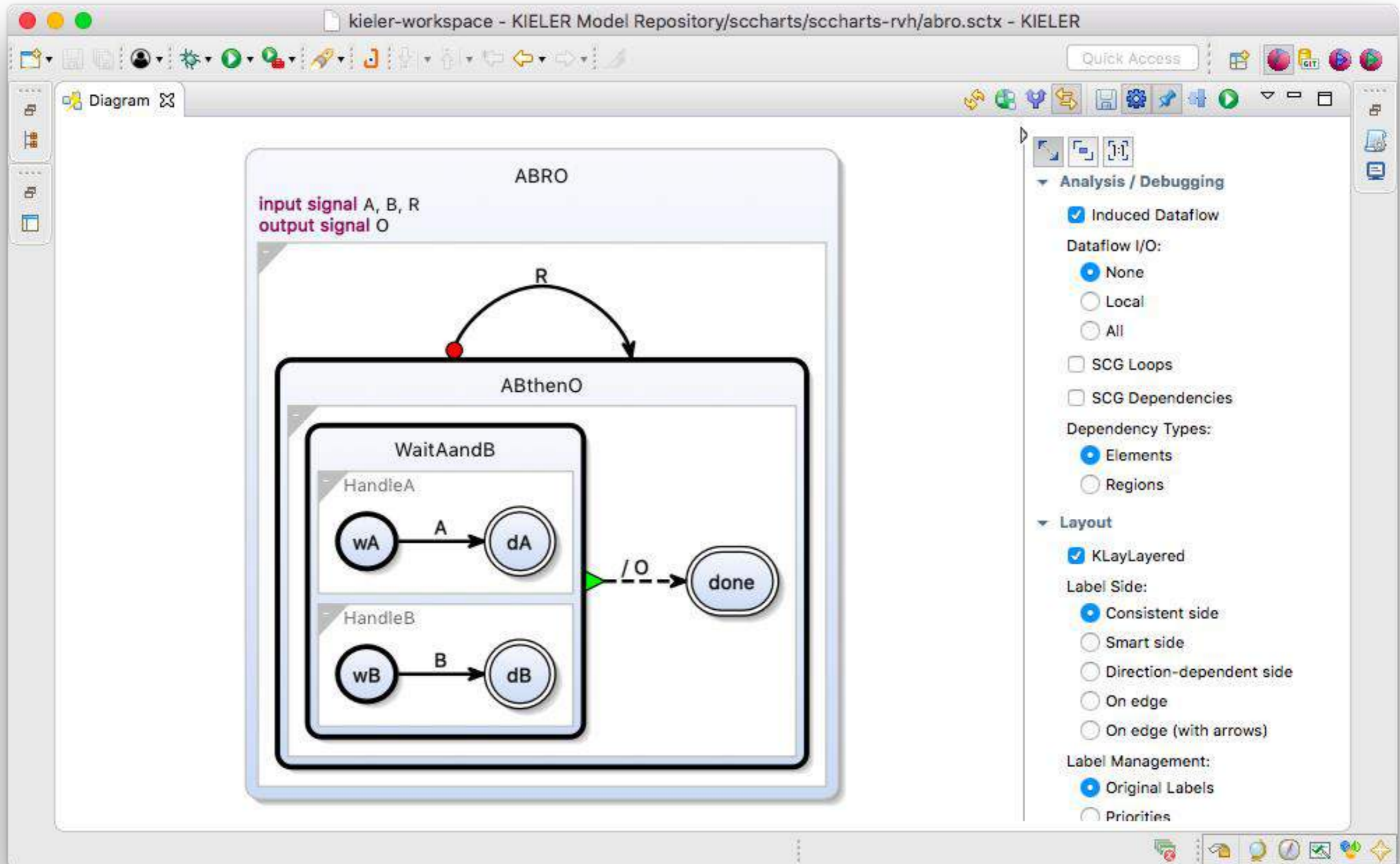


# SCCharts – Classroom-Tested

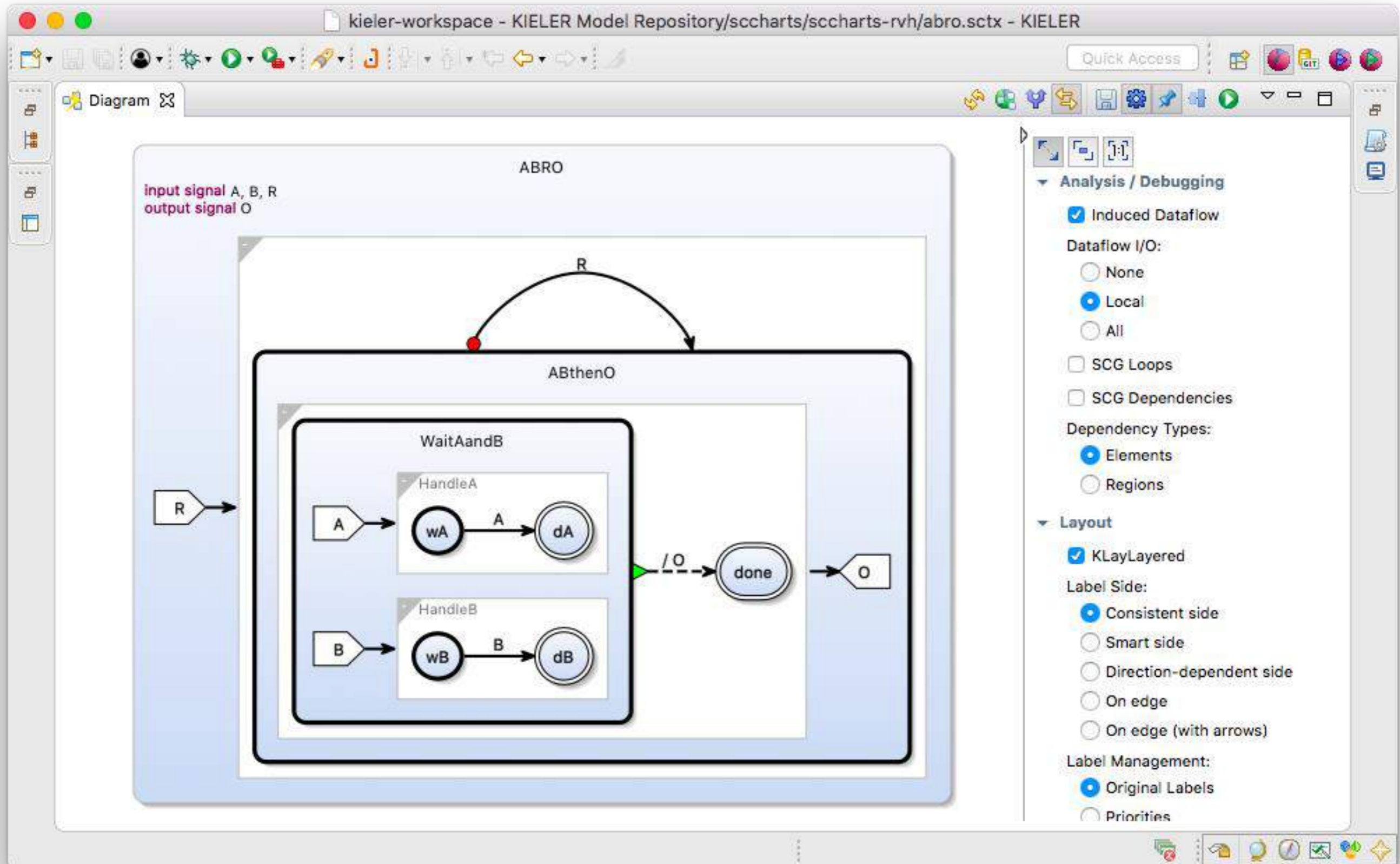
- 10.000 / 135.000 SCChart nodes before/after normalization
- 650.000 lines of C-code
- Compiles in about 2 min's
- 2 ms reaction time

# SCCharts – Modeling Dataflow

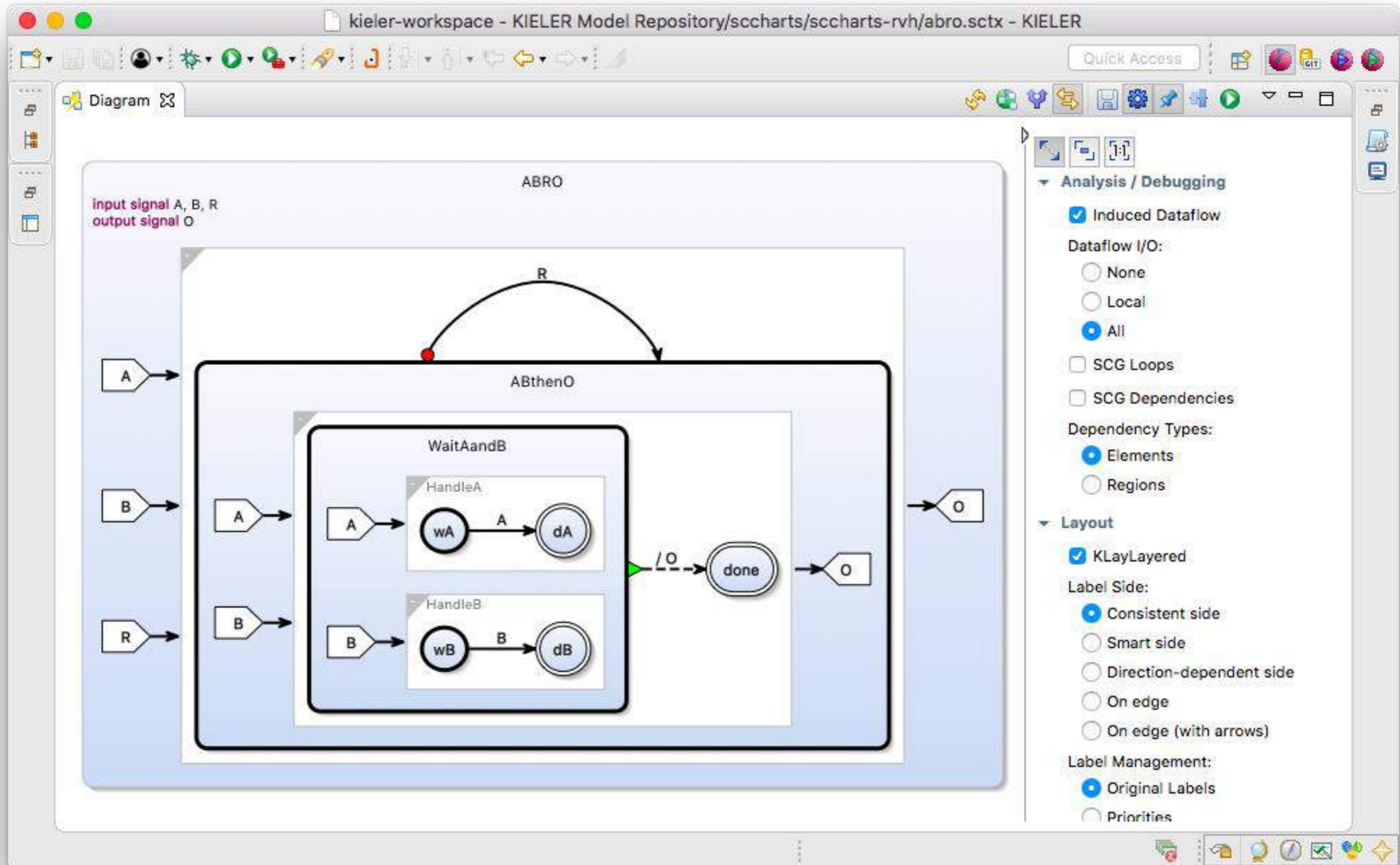
# Dataflow View in ABRO



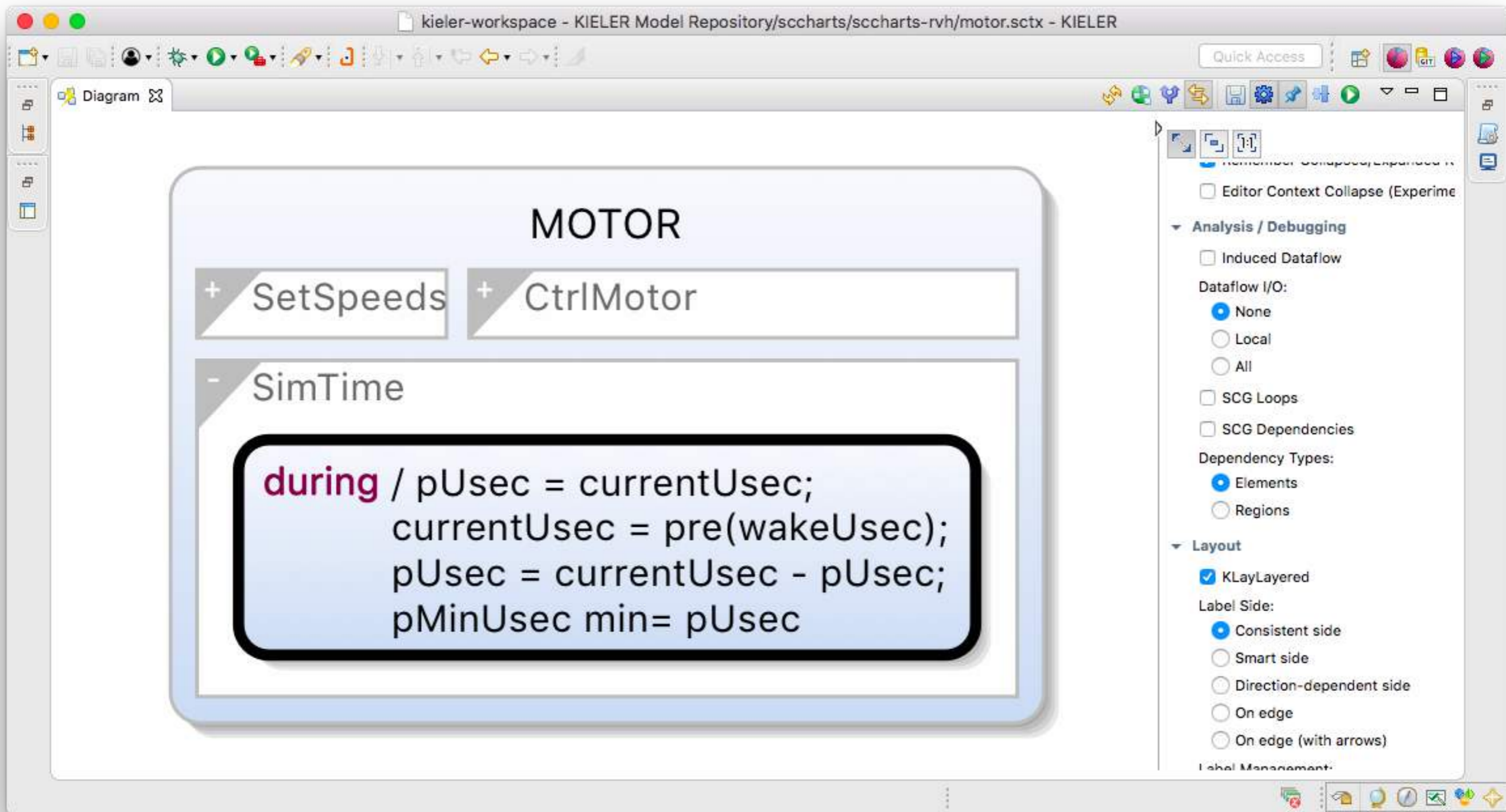
# Dataflow View in ABRO



# Dataflow View in ABRO



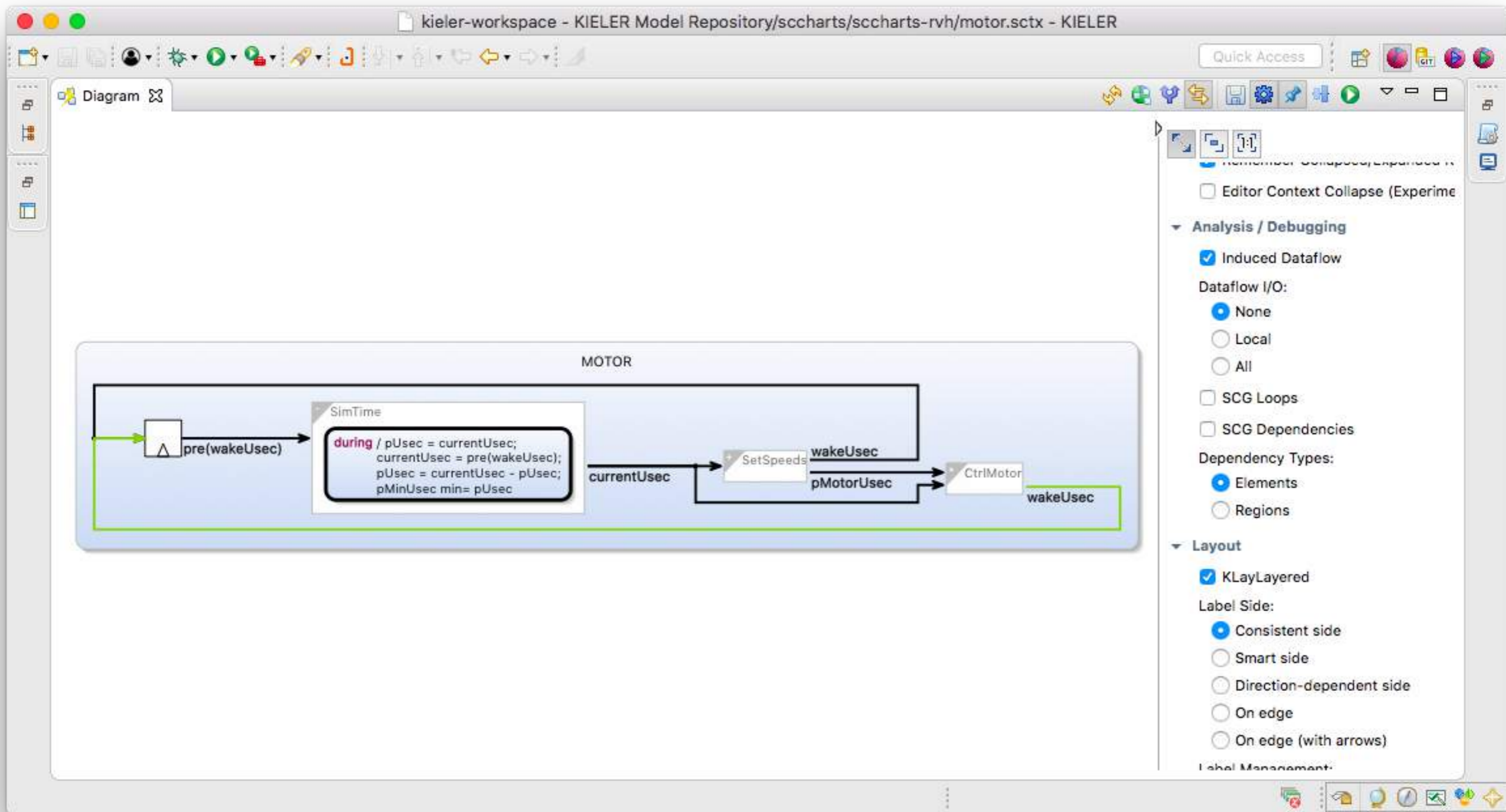
# Dataflow View, with Communication



[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden  
*Augmenting State Models with Data Flow*

Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]

# Dataflow View, with Communication

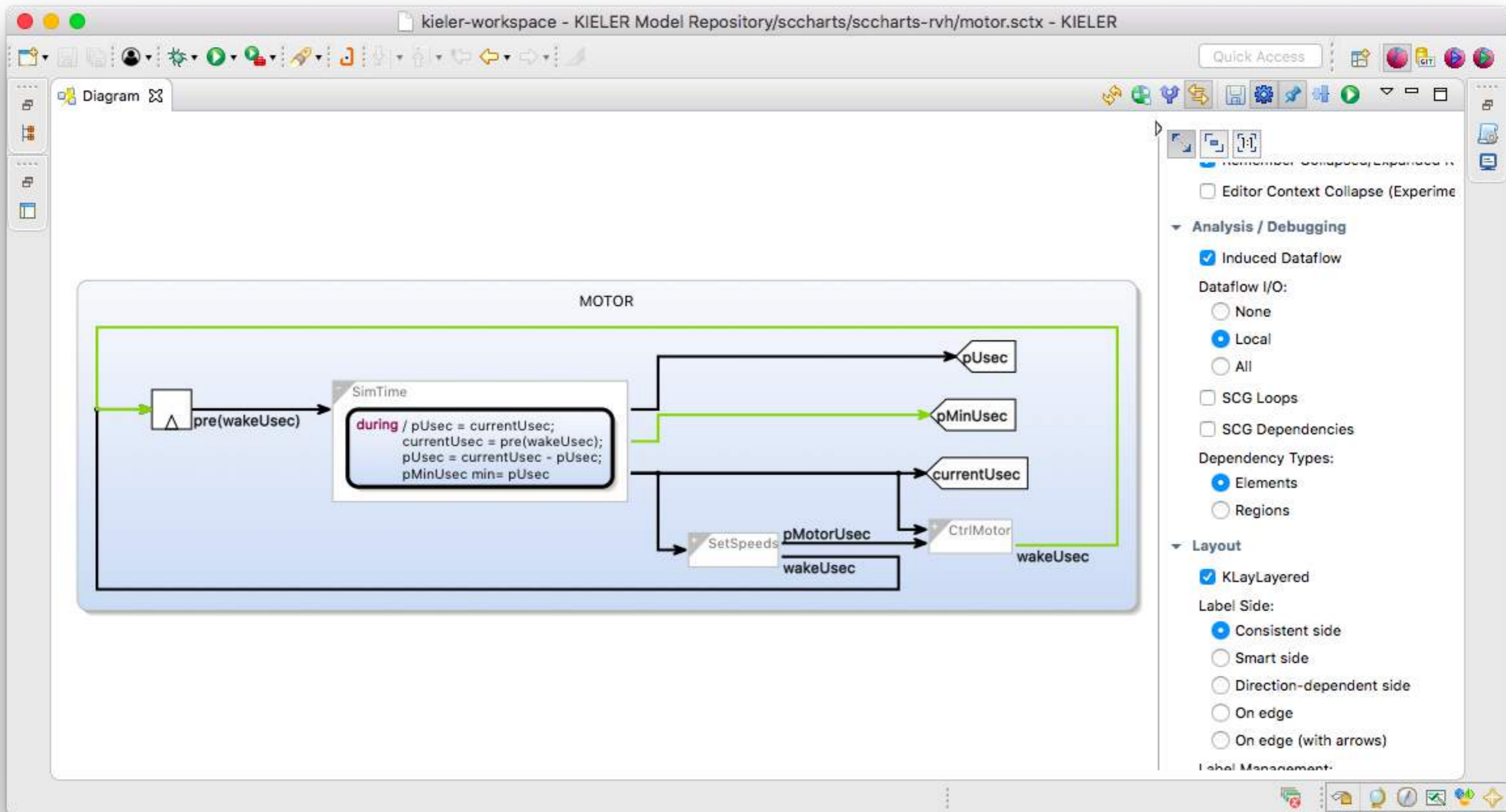


[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden  
*Augmenting State Models with Data Flow*

Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]



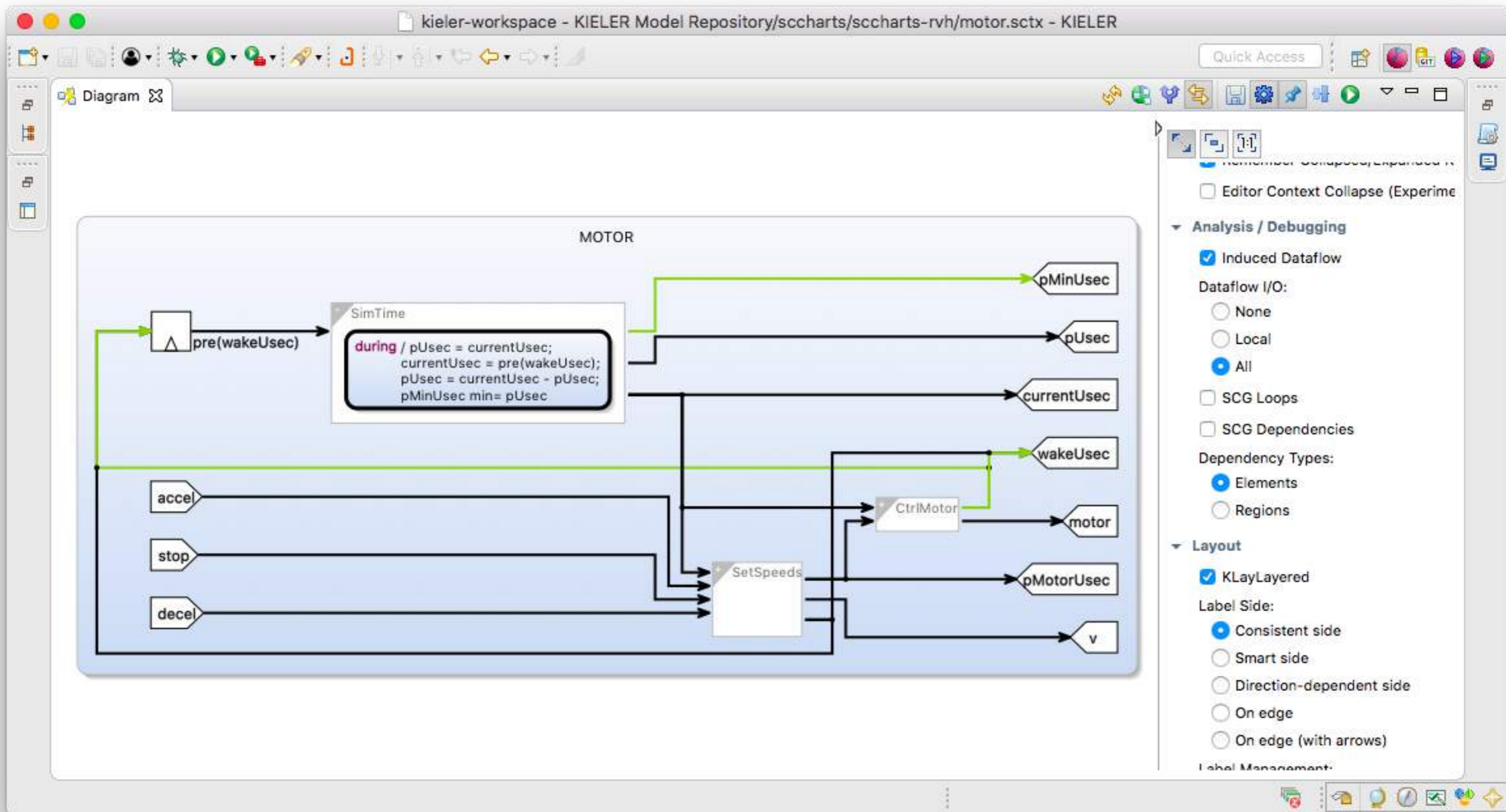
# Dataflow View, with Communication



[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden  
*Augmenting State Models with Data Flow*

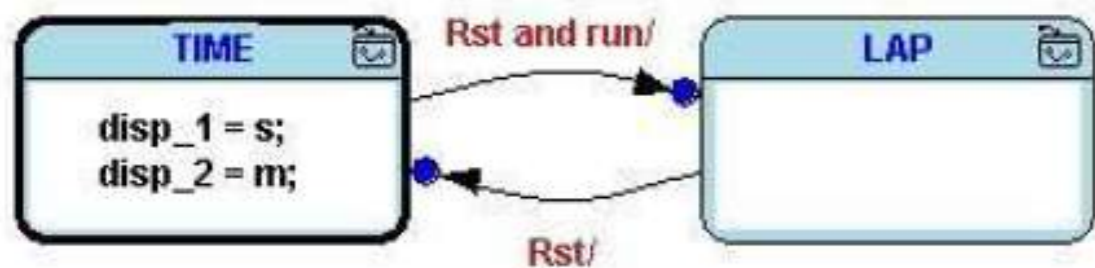
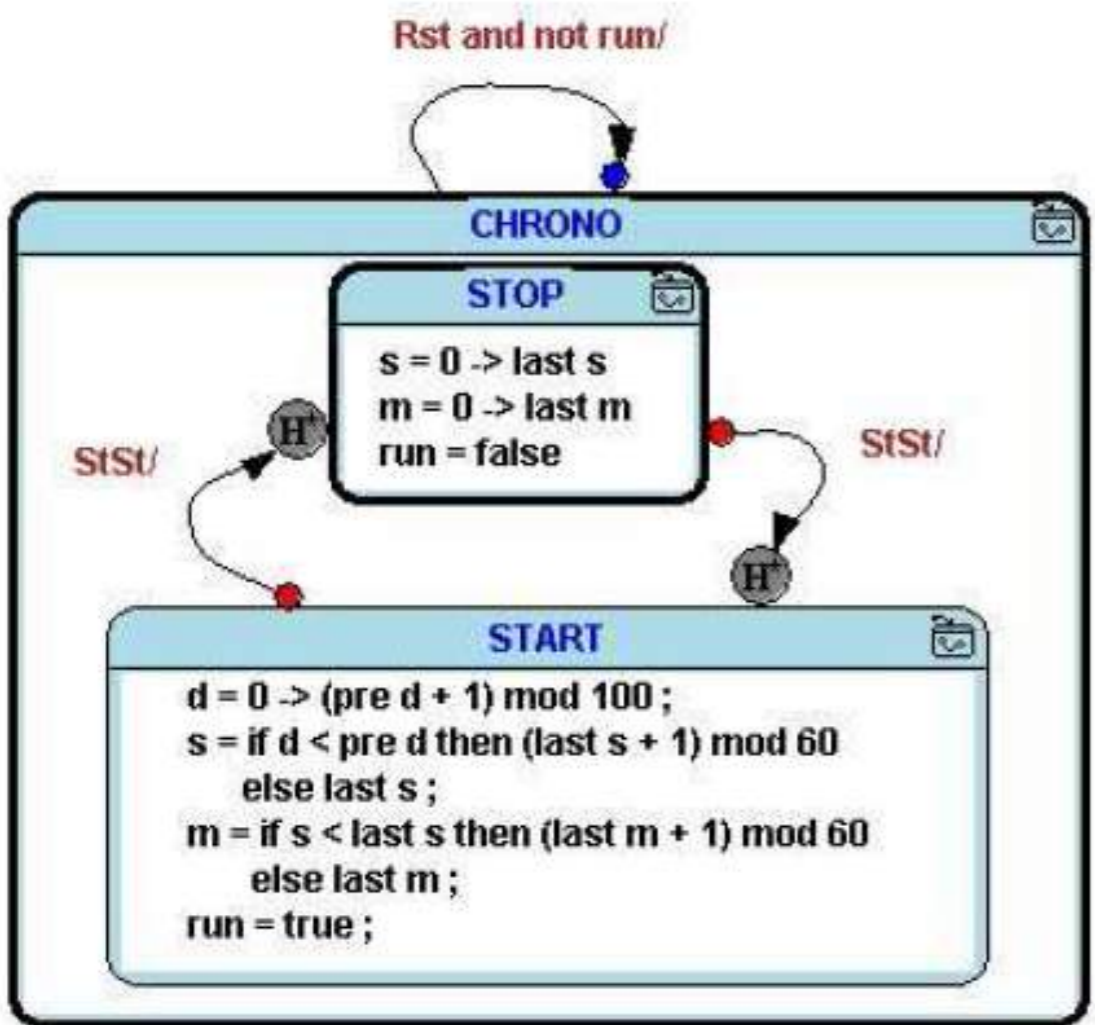
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]

# Dataflow View, with Communication



[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden  
*Augmenting State Models with Data Flow*

Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]



```

let node chrono (StSt, Rst) = (disp_1, disp_2) where
  automaton
  CHRONO ->
  do automaton
    STOP ->
      do s = 0 -> last s
        and m = 0 -> last m
        and run = false
      unless StSt continue START
    | START ->
      let d = 0 -> (pre d + 1) mod 100 in
      do s = if d < pre d
        then (last s + 1) mod 60
        else last s
      and m = if s < last s
        then (last m + 1) mod 60
        else last m
      and run = true
      unless StSt continue STOP
    end
  until Rst and not run then CHRONO
end and
  automaton
  TIME ->
    do disp_1 = s
      and disp_2 = m
    until Rst and run then LAP
  | LAP ->
    do until Rst then TIME
  end
end
  
```

Figure 2: A Chronometer

[Colaço, Pagano, Pouzet,  
*A Conservative Extension of Synchronous Dataflow with State Machines,*  
 EMSOFT'05]

motor.sct chrono.lus

```
1 node chrono (StSt:bool, Rst:bool)
2   returns (disp_1:int, disp_2:int);
3   var s: int;
4   var m: int;
5   var run: bool;
6   var d: int;
7   let
8     automaton
9       CHRONO ->
10        automaton
11          STOP ->
12            s = 0 -> pre s;
13            m = 0 -> pre m;
14            run = false;
15          unless StSt continue START;
16          | START ->
17            d = 0 -> (pre d + 1) mod 100;
18            s = if d < pre d
19              then (pre s + 1) mod 60
20              else pre s;
21            m = if s < pre s
22              then (pre m + 1) mod 60
23              else pre m;
24            run = true;
25          unless StSt continue STOP;
26          end;
27        until Rst and not run then CHRONO;
28      end;
29    automaton
30      TIME ->
31        disp_1 = s;
32        disp_2 = m;
33      until Rst and run then LAP;
34      | LAP ->
35      until Rst then TIME;
36    end;
```

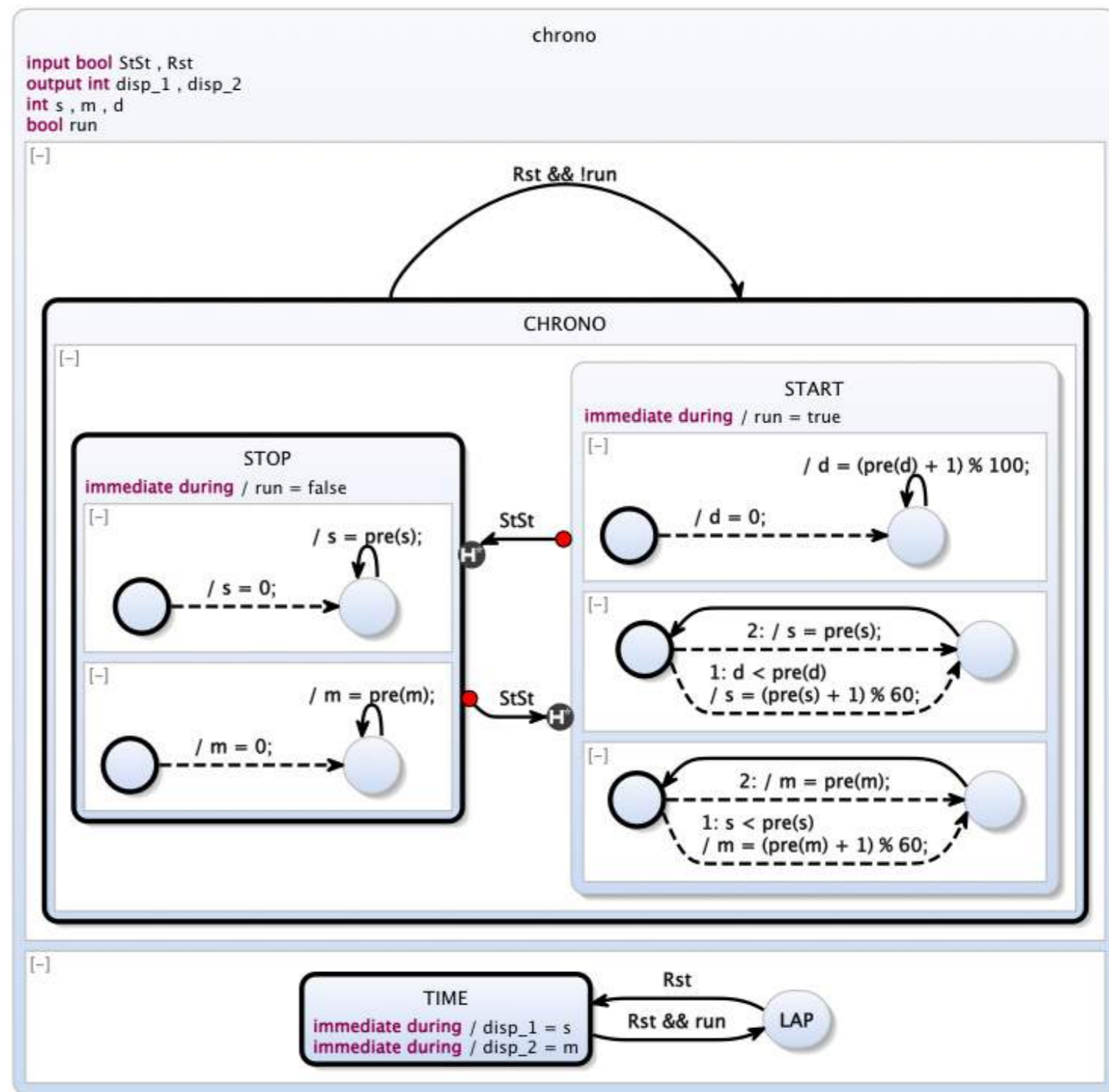
motor.sct chrono.lus

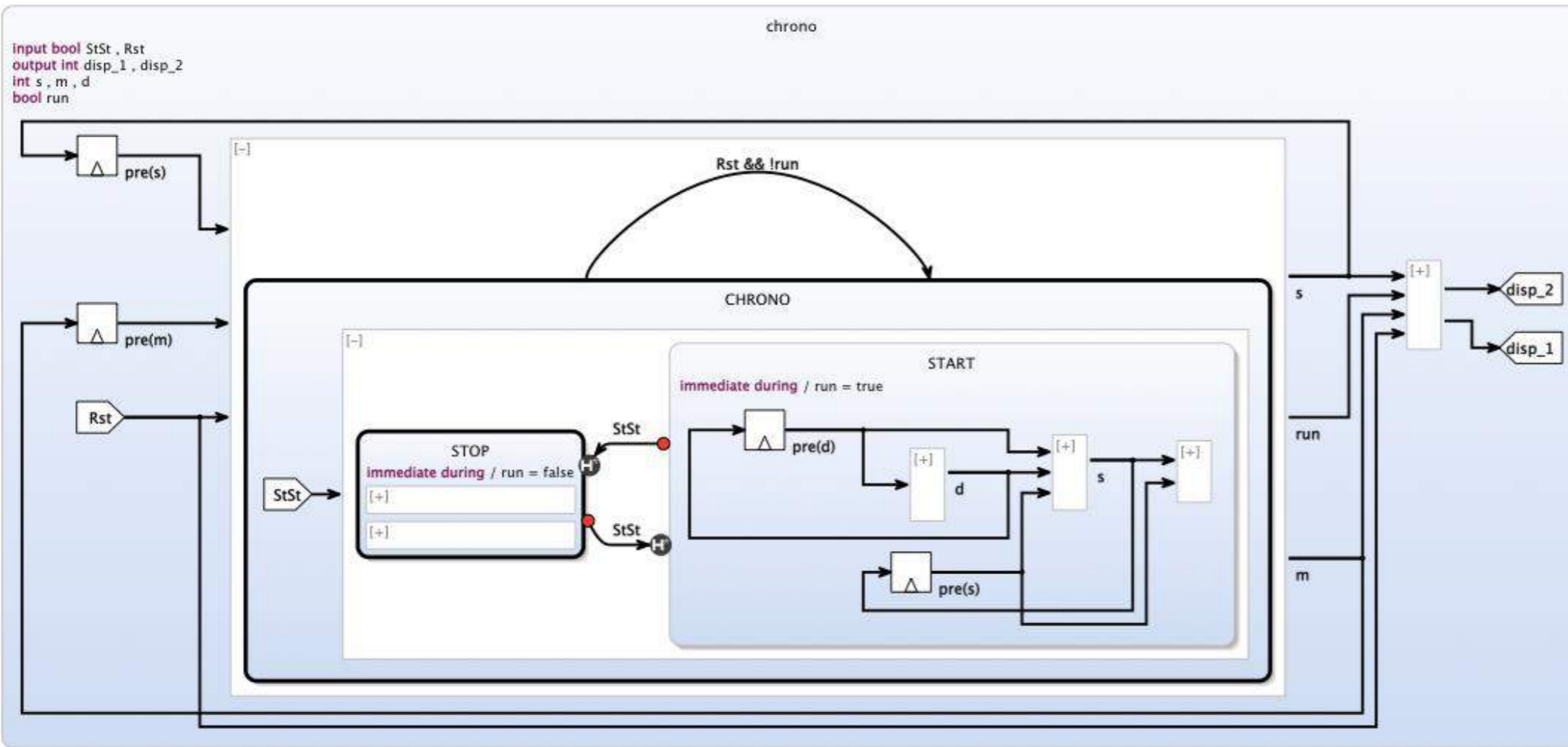
```

1 node chrono (StSt:bool, Rst:bool)
2   returns (disp_1:int, disp_2:int);
3   var s: int;
4   var m: int;
5   var run: bool;
6   var d: int;
7   let
8     automaton
9       CHRONO ->
10        automaton
11          STOP ->
12            s = 0 -> pre s;
13            m = 0 -> pre m;
14            run = false;
15          unless StSt continue START;
16        | START ->
17          d = 0 -> (pre d + 1) mod 100;
18          s = if d < pre d
19              then (pre s + 1) mod 60
20              else pre s;
21          m = if s < pre s
22              then (pre m + 1) mod 60
23              else pre m;
24          run = true;
25          unless StSt continue STOP;
26        end;
27      until Rst and not run then CHRONO;
28    end;
29    automaton
30      TIME ->
31        disp_1 = s;
32        disp_2 = m;
33        until Rst and run then LAP;
34      | LAP ->
35        until Rst then TIME;
36    end;

```

Diagram





- Paper (Black/White)
- Shadow
- Show hidden Elements
- Label Management**
  - Strategy:
    - Original Labels
    - Truncate
    - Semantic Soft Wrapping
    - Priorities
  - Shortening Width: 200
- Initially collapse all regions
- Remember collapse/expand
- Debugging**
  - Show Induced Dataflow
  - Show Dataflow I/O:
    - None
    - Local
    - All
  - Show SCG Dependencies
  - Dependency Types:
    - Elements
    - Regions
- Layout**
  - Direction:
    - HV
    - VH
    - Down
    - Right

# From C to SCCharts

# Extracting Visual Models from C

```

1  int main(int a) {
2      int b = 10, c = 6;
3      if (a > 4) {
4          a = a - 1;
5      } else {
6          a = c + 3;
7      }

```

```

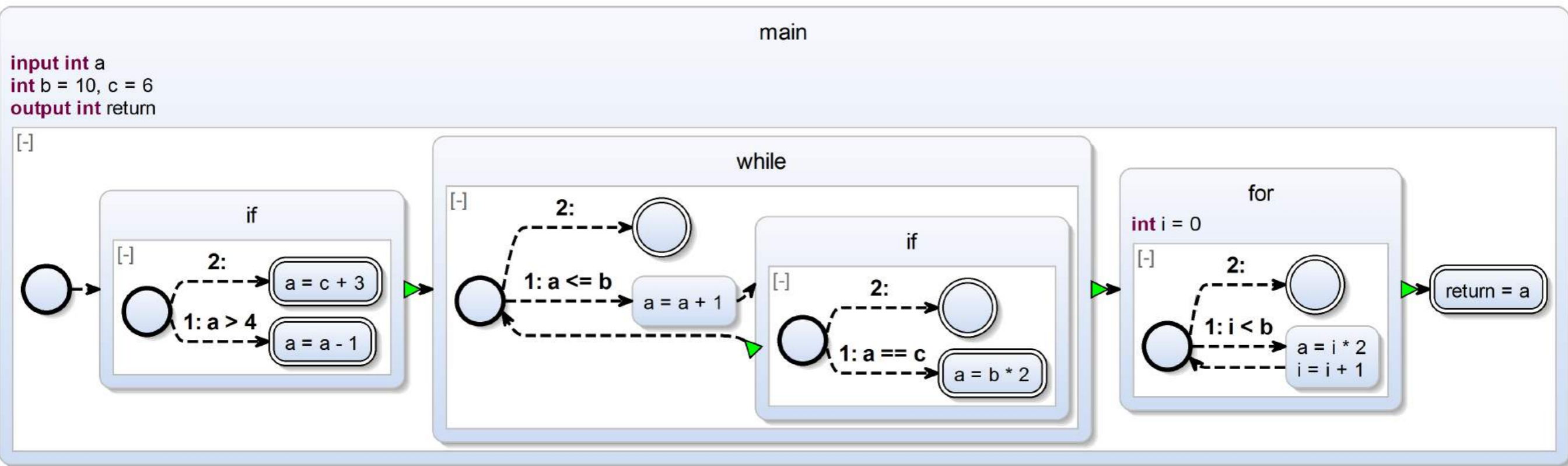
8      while (a <= b) {
9          a = a + 1;
10         if (a == c) {
11             a = b * 2;
12         }
13     }

```

```

14     for (int i = 0; i < b; i = i + 1) {
15         a = i * 2;
16     }
17     return a;
18 }

```



[Smyth, Lenga, von Hanxleden,

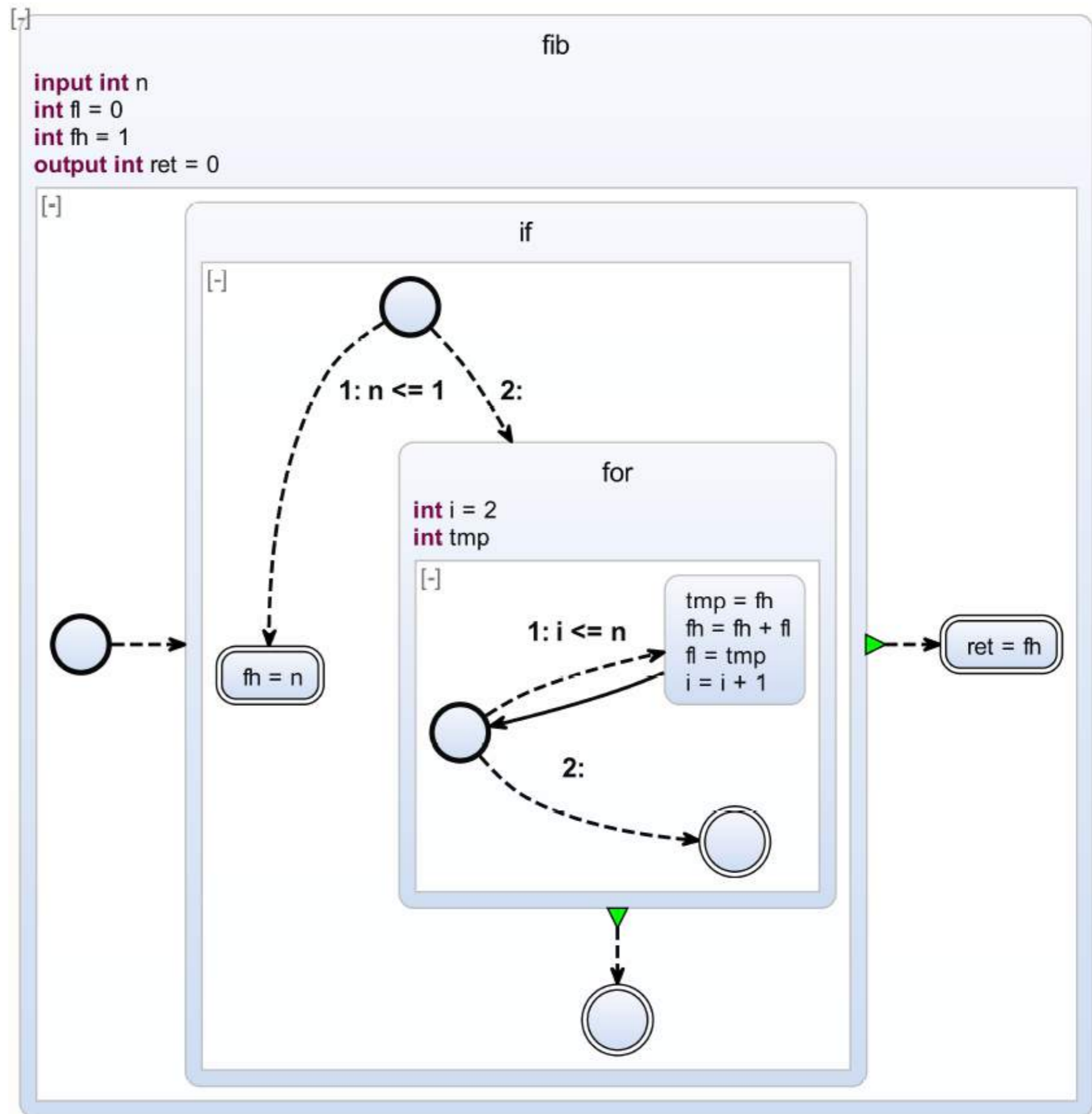
*Model Extraction of Legacy C Code in SCCharts*, ISoLA DS 2016]



```

int fib(int n) {
  int fl = 0, fh = 1;
  if (n <= 1) { fh = n; }
  else {
    for (int i=2; i <= n; i++) {
      int tmp = fh;
      fh += fl;
      fl = tmp;
    }
  }
  return fh;
}

```



```

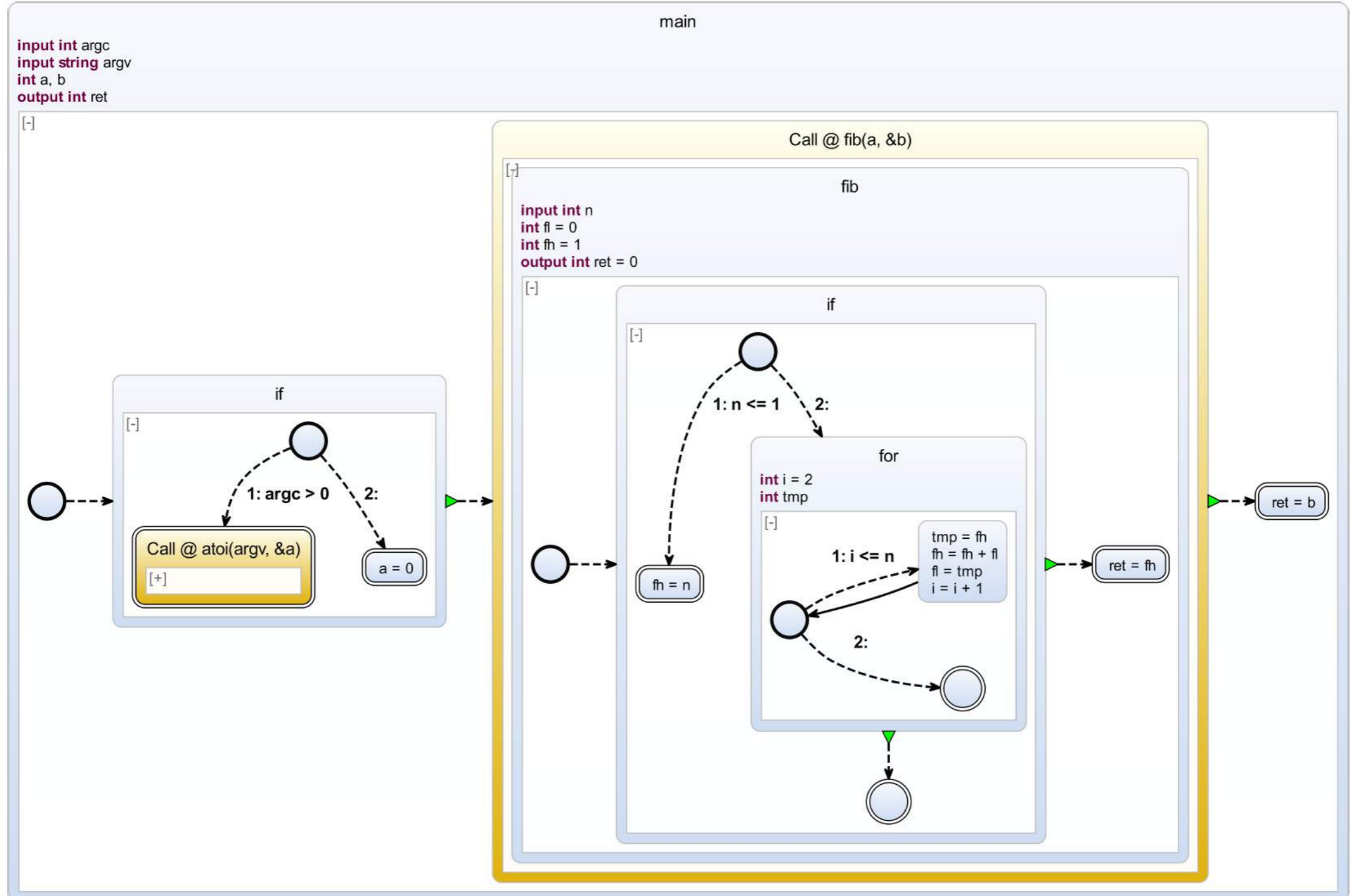
1 int main(int argc, char** argv) {
2     int a, b;
3     if (argc > 0) {
4         a = atoi(argv[0]);
5     } else {
6         a = 0;
7     }
8     b = fib(a);
9     return b;
10 }

```

```

11 int fib(int n) {
12     int fl = 0, fh = 1;
13     if (n <= 1) { fh = n; }
14     else {
15         for (int i=2; i <= n; i++) {
16             int tmp = fh;
17             fh += fl;
18             fl = tmp;

```



```

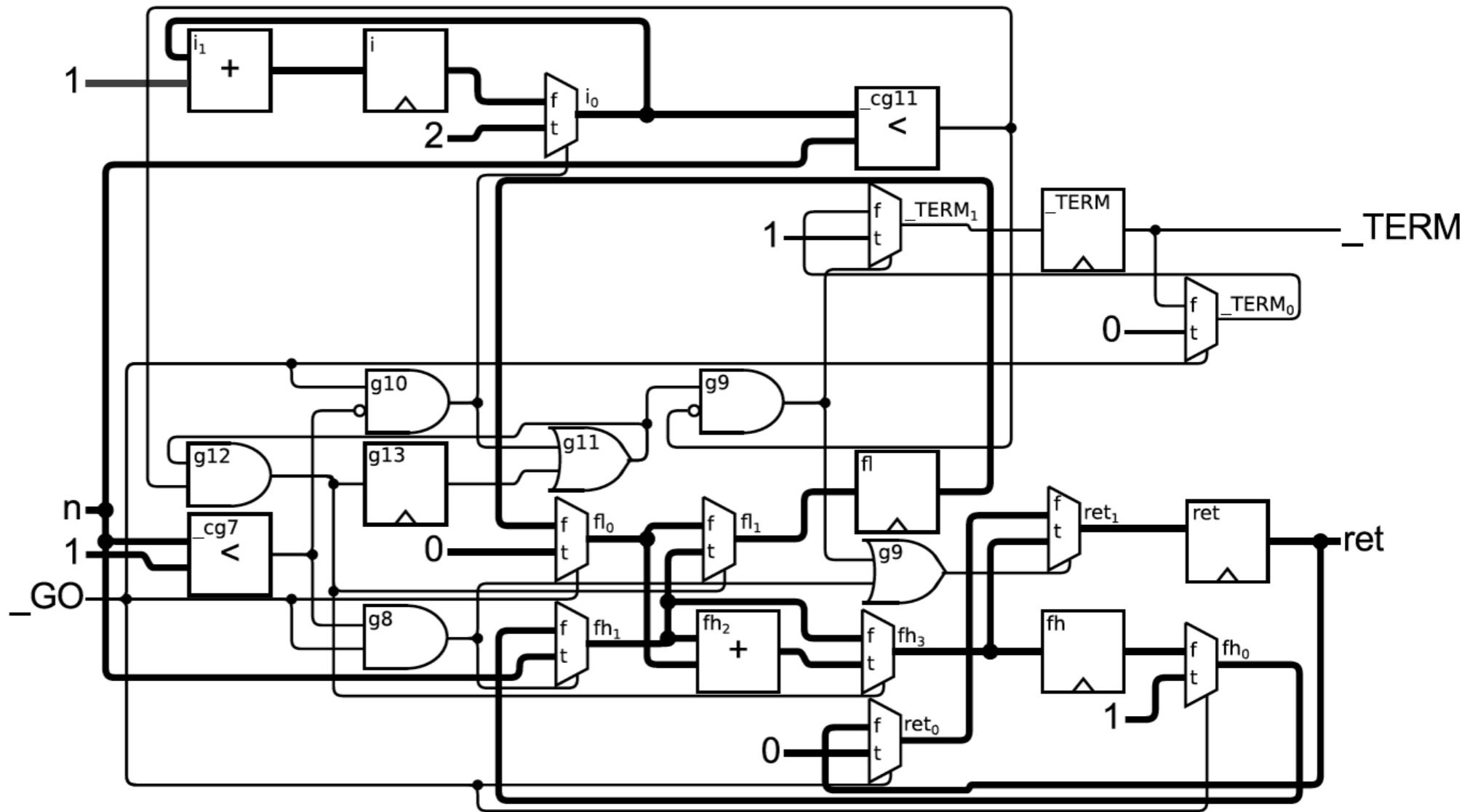
1  typedef struct {
2      char _GO;
3      char g7;
4      ...
5  } TickData1;
6
7  void reset1(TickData1 *d) {
8      d->pg12 = 0;
9      d->_GO = 1;
10     d->_TERM = 0;
11 }
12
13 void tick1(TickData1 *d) {
14     tickLogic1(d);
15     d->_GO = 0;
16     d->pg12 = d->g12;
17 }
18 void tickLogic1(TickData1 *d) {
19     d->g7 = d->_GO;
20     if (d->g7) {
21         d->fl = 0;
22         d->fh = 1;
23     }
24     d->_cg7 = d->n <= 1;
25     d->g8 = d->g7 && d->_cg7;
26     if (d->g8) {
27         d->fh = d->n;
28     }
29     d->g13 = d->pg12;
30     d->g10 = d->g7 && !d->_cg7;
31     if (d->g10) {
32         d->_fib_int_local_i = 2;
33     }
34     d->g11 = d->g13 || d->g10;
35     d->_cg11 =
36         d->_fib_int_local_i <= d->n;
37     d->g12 = d->g11 && d->_cg11;
38     if (d->g12) {
39         d->_fib_int_local_tmp = d->fh;
40         d->fh = d->fh + d->fl;
41         d->fl = d->_fib_int_local_tmp;
42         d->_fib_int_local_i =
43             d->_fib_int_local_i + 1;
44     }
45     d->g9 = d->g11 &&
46         !d->_cg11 || d->g8;
47     if (d->g9) {
48         d->ret = d->fh;
49         d->_TERM = 1;
50     }
51 }

```

```

1  typedef struct {
2      char _GO;
3      char g7;
4      ...
5  } TickData1;
6
7  void reset1(TickData1 *d) {
8      d->pg12 = 0;
9      d->_GO = 1;
10     d->_TERM = 0;
11 }
12
13 void tick1(TickData1 *d) {
14     tickLogic1(d);
15     d->_GO = 0;
16     d->pg12 = d->g12;
17 }
18 void tickLogic1(TickData1 *d) {
19     d->g7 = d->_GO;
20     if (d->g7) {
21         d->fl = 0;
22         d->fh = 1;
23     }
24     d->_cg7 = d->n <= 1;
25     d->g8 = d->g7 && d->_cg7;
26     if (d->g8) {
27         d->fh = d->n;
28     }
29     d->g13 = d->pg12;
30     d->g10 = d->g7 && !d->_cg7;
31     if (d->g10) {
32         d->_fib_int_local_i = 2;
33     }
34     d->g11 = d->g13 || d->g10;
35     d->_cg11 =
36         d->_fib_int_local_i <= d->n;
37     d->g12 = d->g11 && d->_cg11;
38     if (d->g12) {
39         d->_fib_int_local_tmp = d->fh;
40         d->fh = d->fh + d->fl;
41         d->fl = d->_fib_int_local_tmp;
42         d->_fib_int_local_i =
43             d->_fib_int_local_i + 1;
44     }
45     d->g9 = d->g11 &&
46         !d->_cg11 || d->g8;
47     if (d->g9) {
48         d->ret = d->fh;
49         d->_TERM = 1;
50     }
51 }

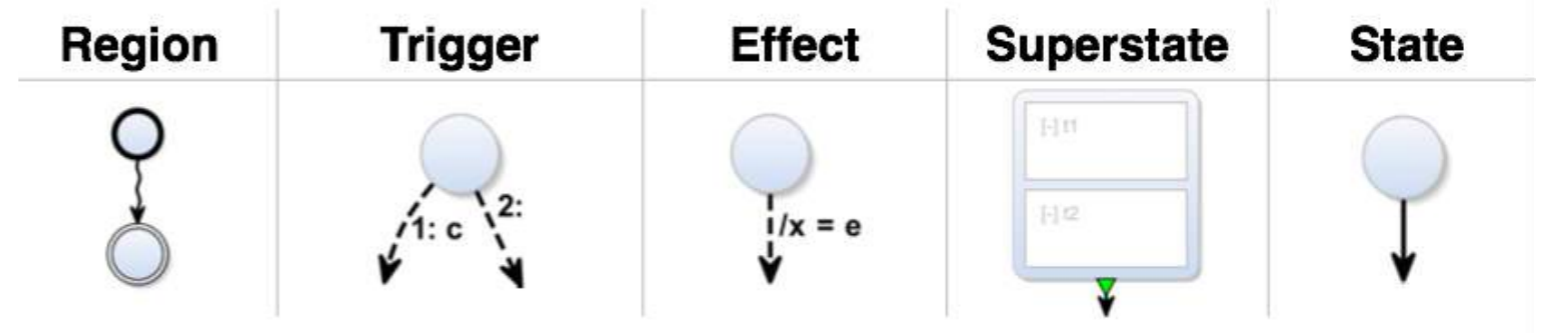
```



# SCCharts Wrap-Up

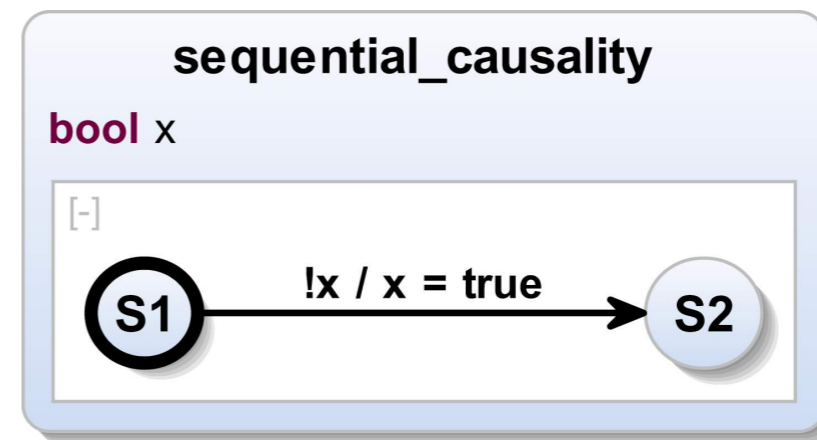
## Language

- 5 core constructs
- Smörgåsbord of extensions



## Model of Computation

- Relaxed synchrony
- Still determinate
- Can model C programs



## Compilation

- M2M transformations
- Stress-tested in KIELER



Still plenty of things to do: Variants on SC MoC, optimize code generation, pragmatics improvements for schedulability analysis, ...

# Code Generation

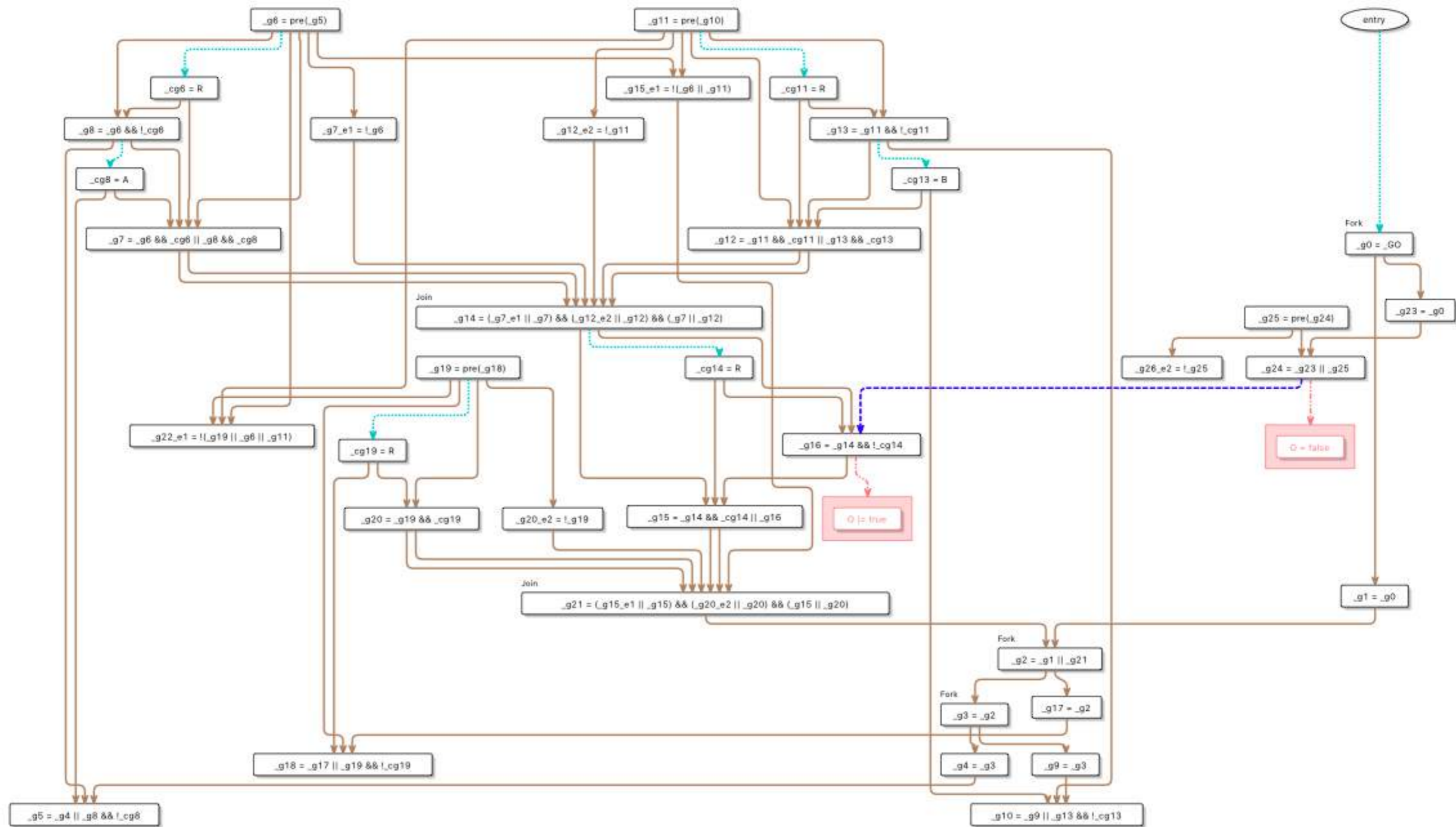
# Downstream Compilation

So far, two alternative compilation strategies from SCL/SCG to C/VHDL

	Dataflow	Priority
Accepts instantaneous loops	—	+
Can synthesize hardware	+	—
Can synthesize software	+	+
Size scales well (linear in size of SCChart)	+	+
Speed scales well (execute only active parts)	—	+
Instruction-cache friendly (good locality)	+	—
Pipeline friendly (little/no branching)	+	—
WCRT predictable (simple control flow)	+	+/-
Low execution time jitter (simple/fixed flow)	+	—

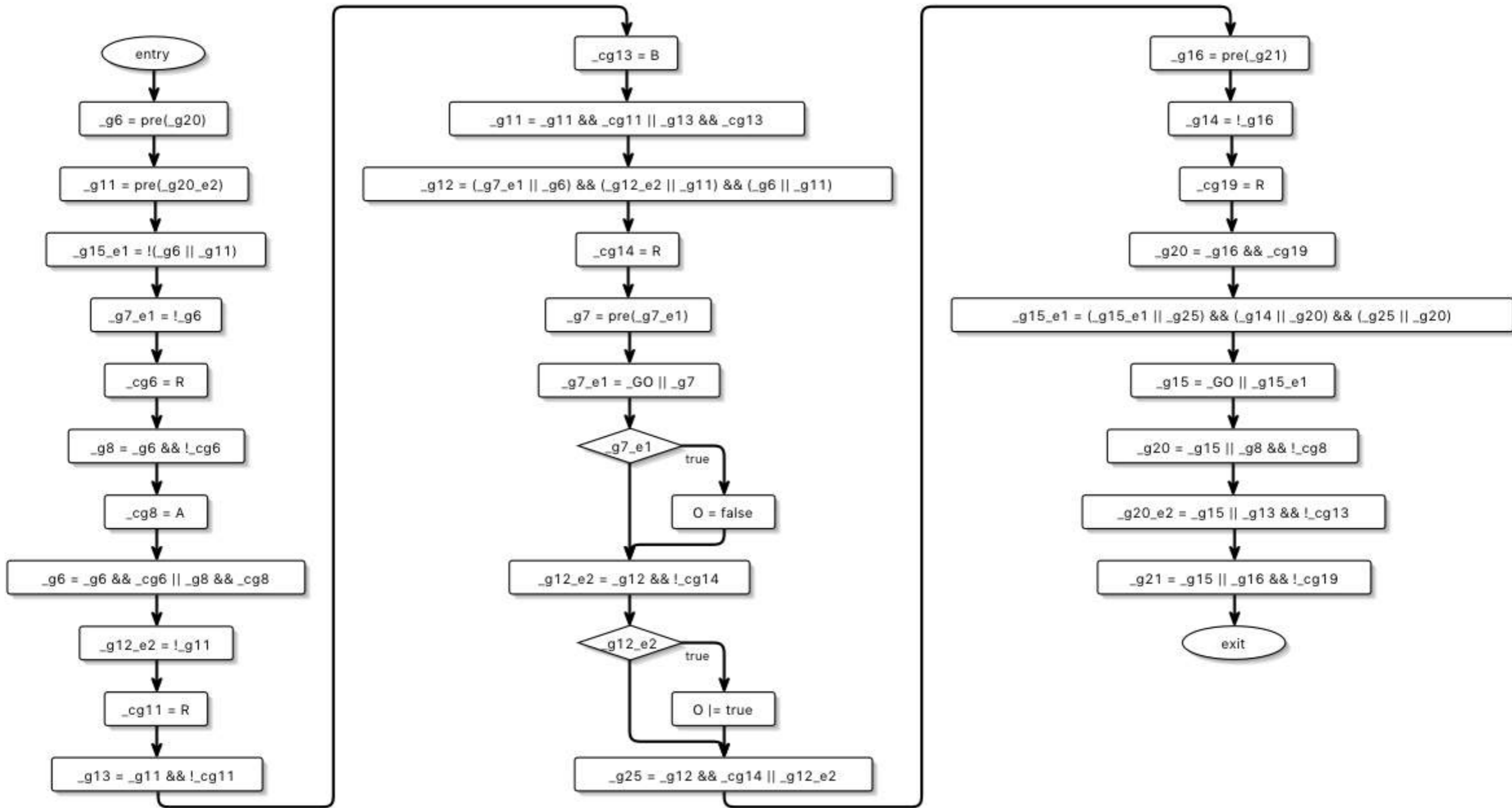
[von Hanxleden, Duderstadt, Motika, et al.,  
*SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications*,  
PLDI'14]

# Compilation Option 1: Dataflow

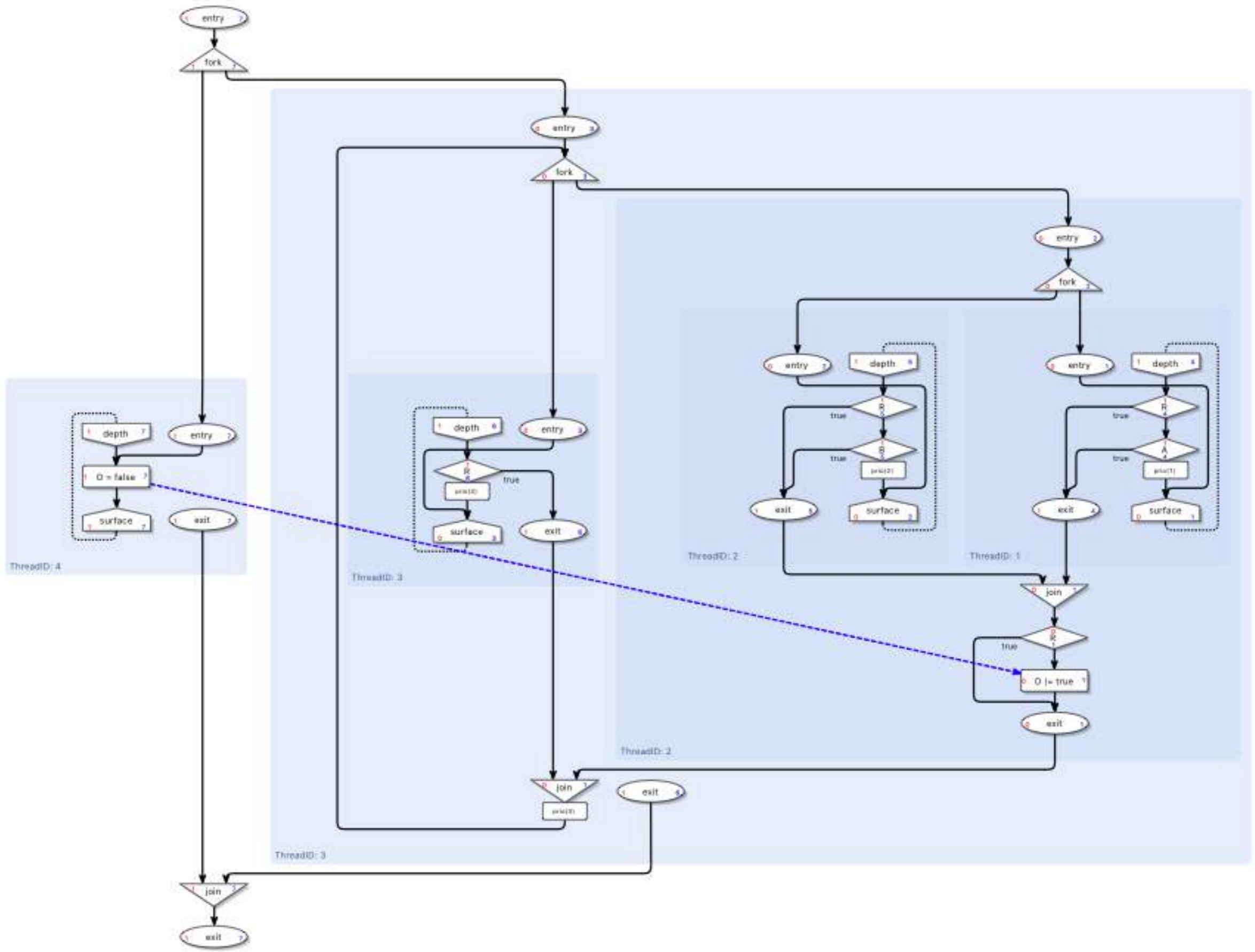




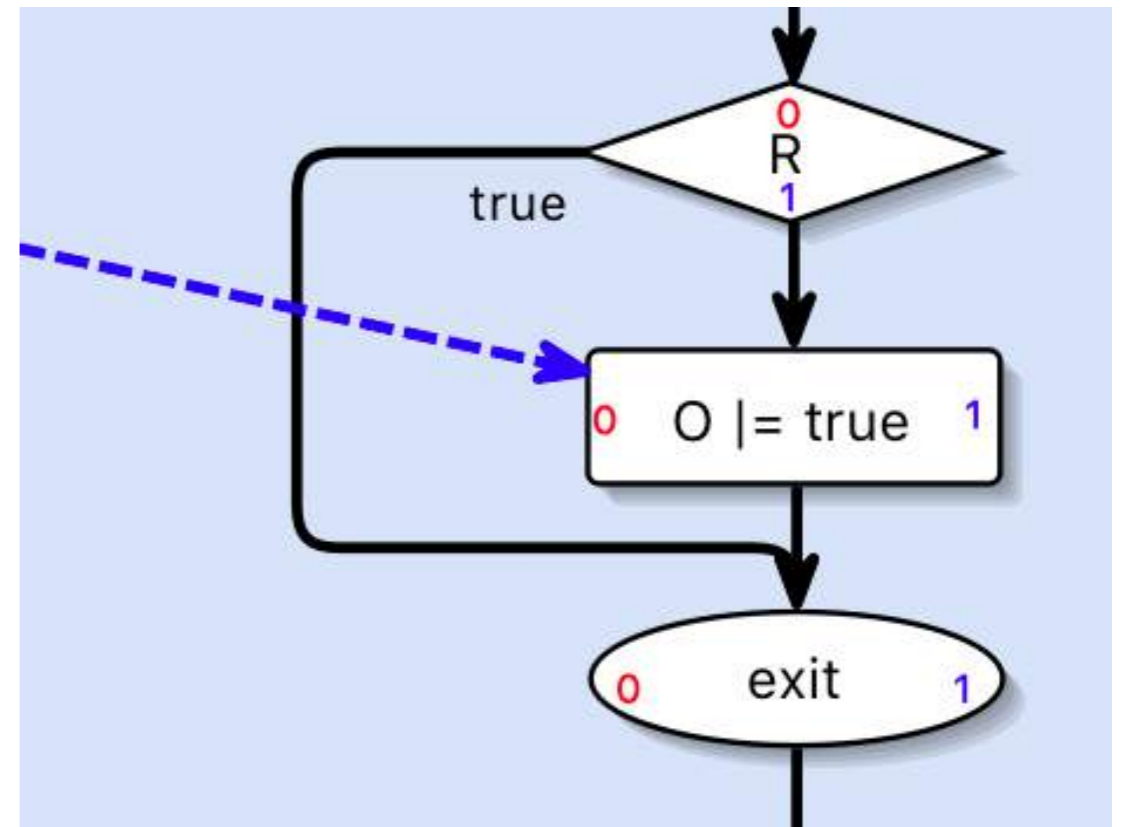
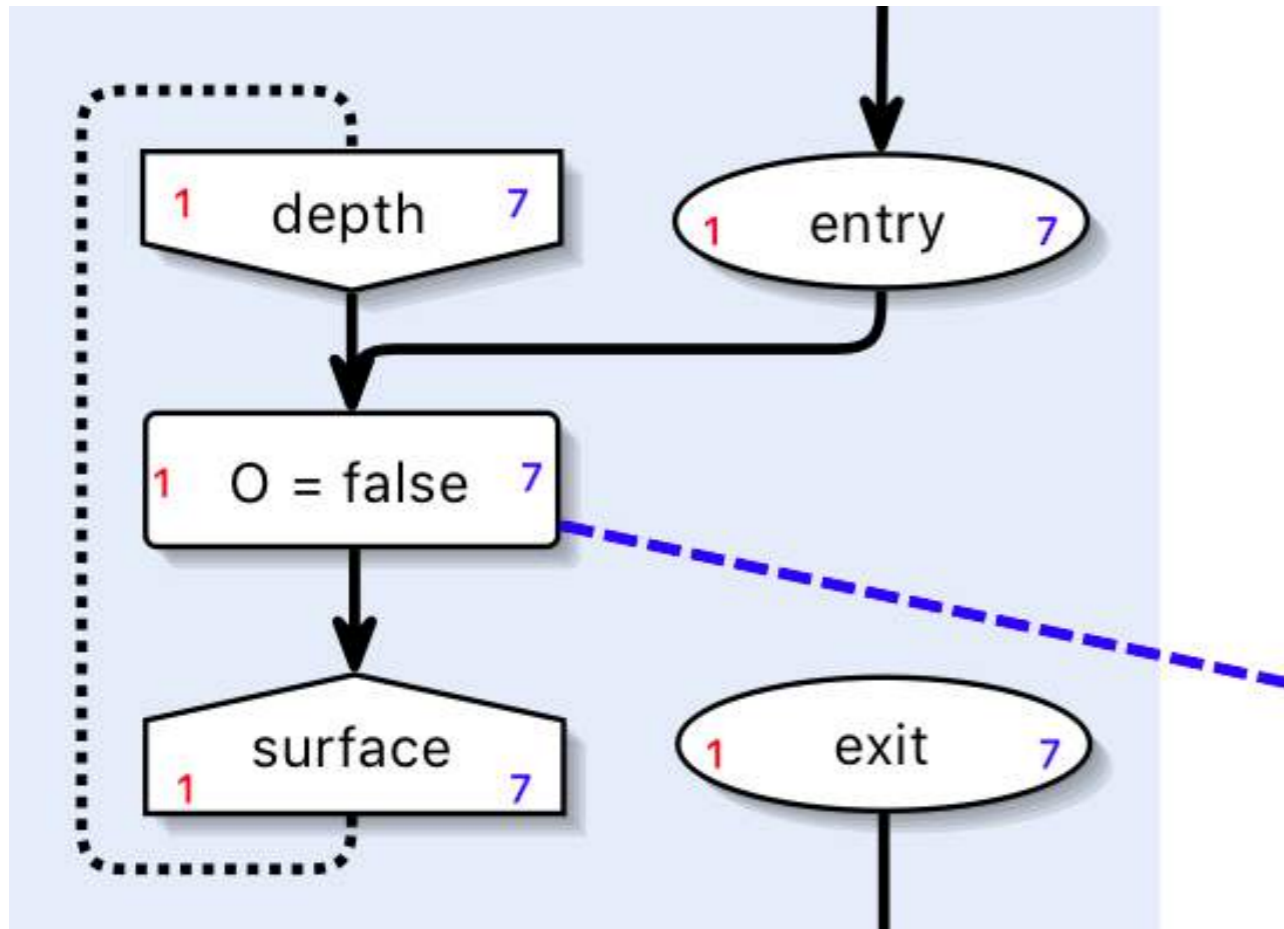
# Dataflow SCG



# Compilation Option 2: Priority-Based



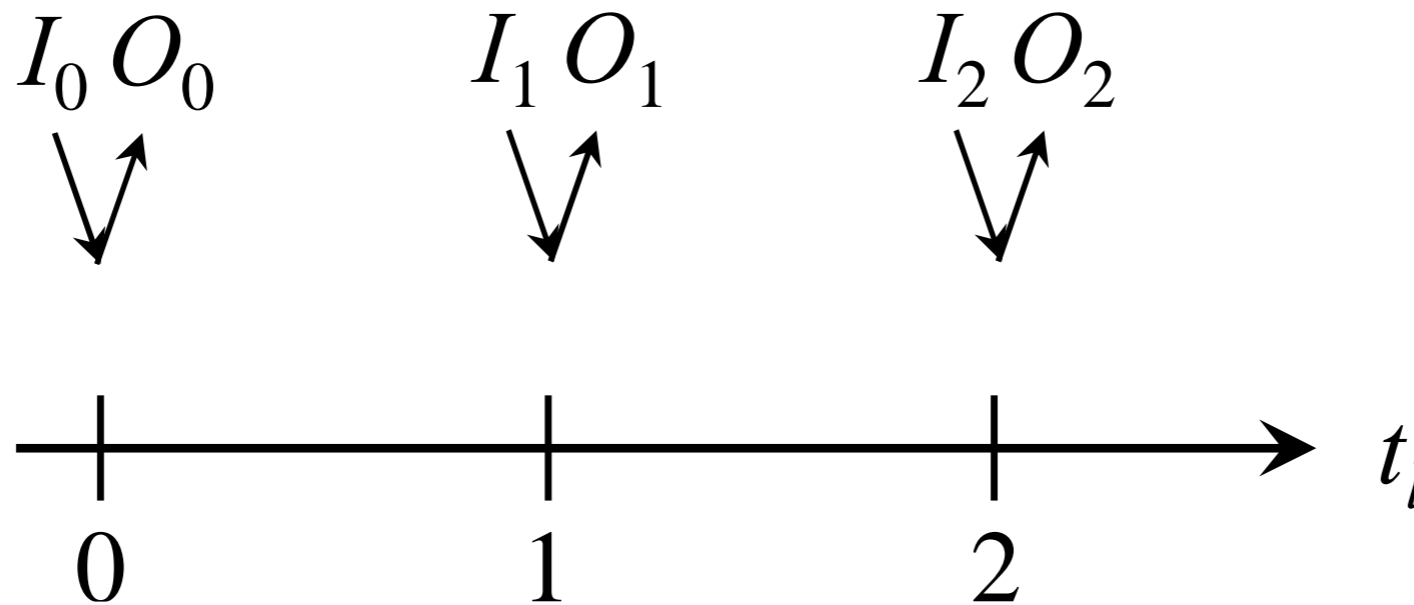
# Compilation Option 2: Priority-Based



# Priority-Based C Code

```
int tick() {  
  
    tickstart(7);  
    fork1(_region_0,_region_1, 3) {  
        _region_0:  
        O = 0;  
        pause;  
        goto _region_0;  
  
    } par {  
        _region_1:  
        fork1(_region_2,_region_3, 2) {  
            _region_2:  
            prio(6);  
            pause;  
            if(R){  
  
            } else {  
                prio(3);  
                goto _region_2;  
            }  
  
        } par {  
            HandleA:  
            prio(4);  
            pause;  
            if(R){  
  
            } else {  
                if(A){  
  
                } else {  
                    prio(2);  
                    goto HandleB;  
                }  
            }  
  
            } par {  
                _region_3:  
                fork1(HandleB,HandleA, 1) {  
                    HandleB:  
                    prio(5);  
                    pause;  
                    if(R){  
  
                    } else {  
                        if(B){  
  
                        } else {  
                            prio(1);  
                            goto HandleA;  
                        }  
                    }  
  
                } join2(2, 5);  
                if(R){  
  
                } else {  
                    O |= 1;  
                }  
  
            } join2(3, 6);  
            prio(3);  
            goto _region_1;  
        } join1(7);  
        tickreturn();  
    }  
}
```

# SCCharts and Time



- Synchrony Hypothesis:  
Outputs are synchronous with inputs
- Computation "does not take time"
- Actual computation time does not influence result
- Sequence of outputs **determined** by inputs

# Synchronous Execution

```
Initialize Memory
for each input event do
  Compute Outputs
  Update Memory
end
```

```
Initialize Memory
for each clock tick do
  Read Inputs
  Compute Outputs
  Update Memory
end
```

**Fig. 1** Two common synchronous execution schemes: event driven (left) and sample driven (right).

[Benveniste et al., *The Synchronous Languages Twelve Years Later*, Proc. IEEE, 2003]

# Multiform Notion of Time

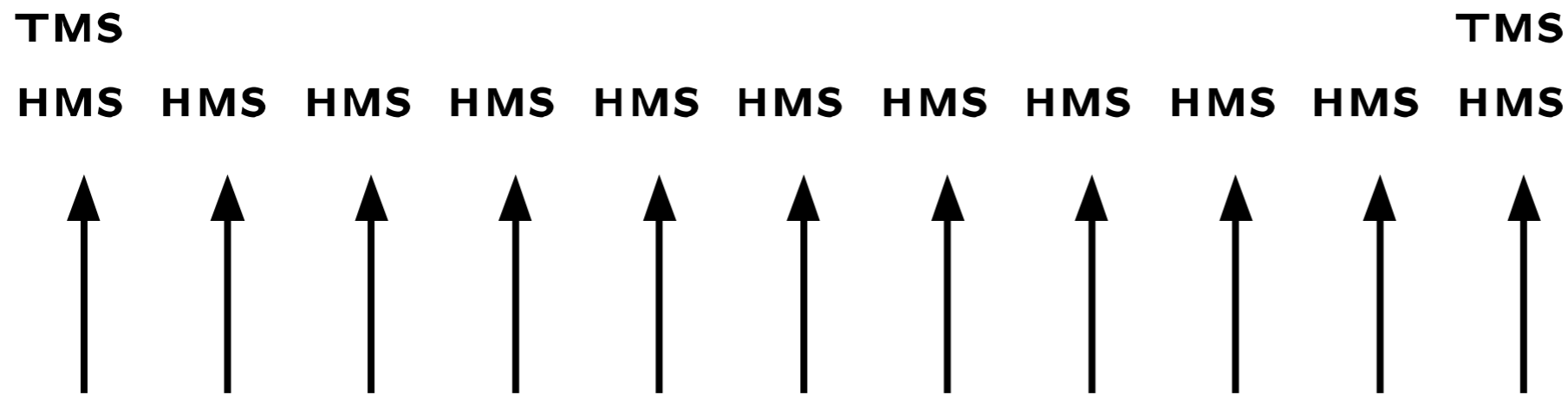
*Only the simultaneity and precedence of events are considered.*

*This means that the physical time does not play any special role.*

*This is called multiform notion of time.*

[<https://en.wikipedia.org/wiki/Esterel>]

# Packaging Physical Time as Events



[Timothy Bourke, SYNCHRON 2009]

Event "HMS": 100  $\mu$ sec have passed since last HMS

Event "TMS": 1000  $\mu$ sec have passed since last TMS



# A Problem With That ...

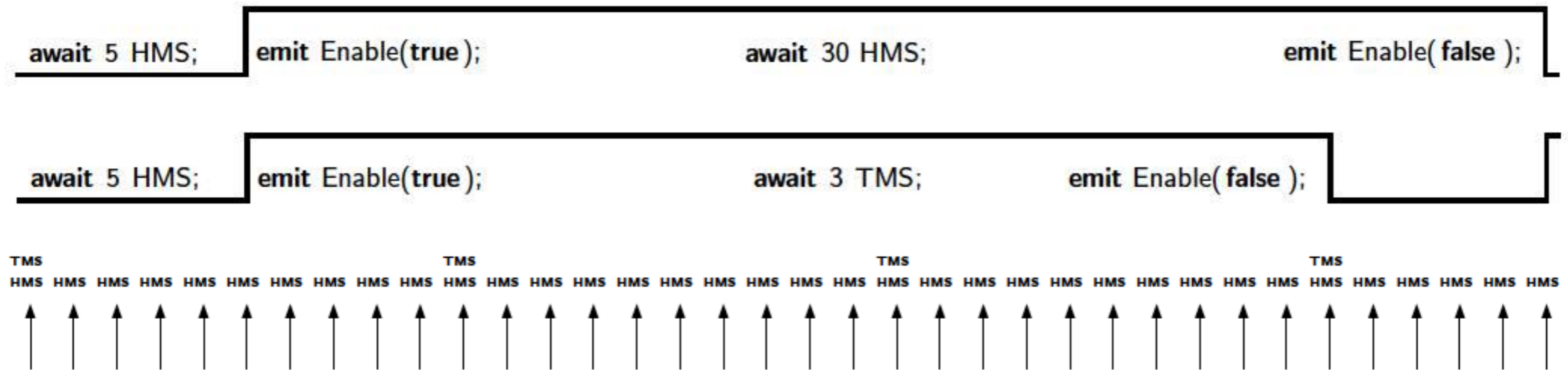
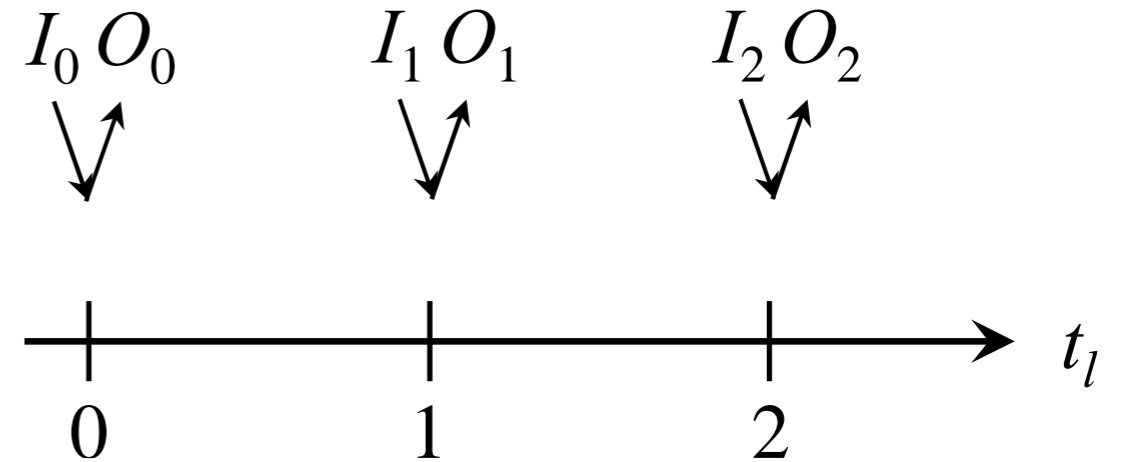


Fig. 4: Granularity of timing inputs

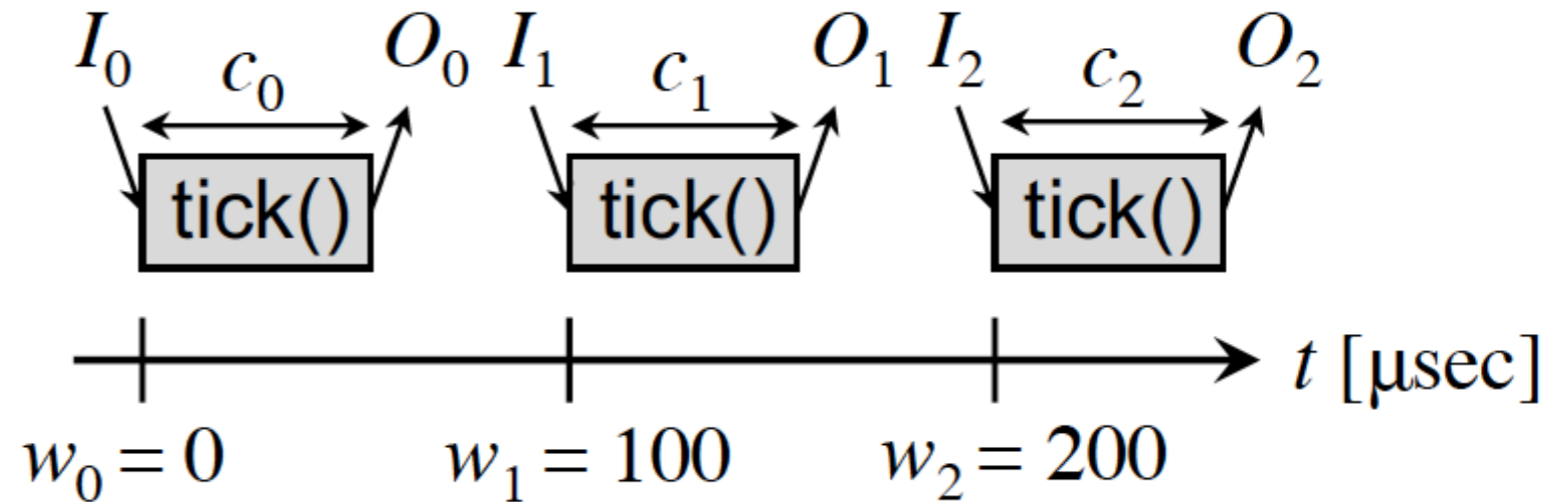
[Timothy Bourke, SYNCHRON 2009]

# Dynamic Ticks

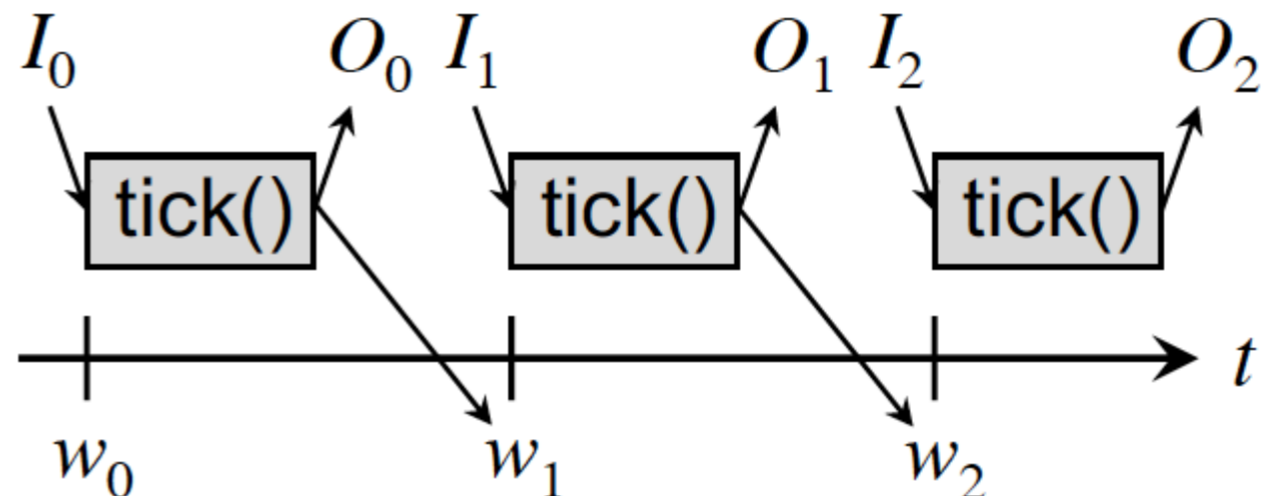
- Recall logical time:



- Physical time, time-triggered:



- Physical time, dynamic ticks:



[von Hanxleden, Bourke, Girault,

*Real-Time Ticks for Synchronous Programming*, FDL'17]

# SCCharts Textual Representation

```
/** Controller for stepper motor
*/
```

```
scchart MOTOR {
```

```
output int currentUsec = 0; // [usec] Current simulated time;
// when deployed, this should be input
```

```
output int wakeUsec; // [usec] Time for next wake-up
```

```
input bool accel, decel; // Increase/decrease speed
```

```
input bool stop; // Emergency stop - sets (angular) speeds to 0
```

```
output bool motor = false; // Motor pulse
```

```
output float v; // [cm/sec] Robot speed
```

```
output int pMotorUsec; // [usec] Half period for motor
```

```
int pSetSpeedsMaxUsec = 500000; // [usec] Maximum period of speed control loop
```

```
int pSetSpeedsMinUsec = 400000; // [usec] Minimum period of speed control loop
```

```
output int pUsec; // [usec] Previous period (delta of wake-up times)
```

```
output int pMinUsec = pSetSpeedsMaxUsec; // [usec] Minimum period
```

```
float dV = 2; // [cm/sec] Delta v applied during one speed
// control loop cycle
```

```
float vMax = 20; // [cm/sec] Max speed of left/right motor
```

```
float cmPerHalfPeriod = 1; // [cm] Distance traveled by motor per half period
// (duration of true or false)
```

```

/** Controller for stepper motor
 */

schart MOTOR {
output int currentUsec = 0; // [uSec] Current simulated time; when deployed, this should be input
output int wakeUsec; // [uSec] Time for next wake-up

input bool accel, decel; // Increase/decrease speed
input bool stop; // Emergency stop - sets (angular) speeds to 0

output bool motor = false; // Motor pulse
output float v; // [cm/sec] Robot speed
output int pMotorUsec; // [uSec] Half period for motor

int pSetSpeedsMaxUsec = 500000; // [uSec] Maximum period of speed control loop
int pSetSpeedsMinUsec = 400000; // [uSec] Minimum period of speed control loop
output int pUsec; // [uSec] Previous period (delta of wake-up times)
output int pMinUsec = pSetSpeedsMaxUsec; // [uSec] Minimum period

float dV = 2; // [cm/sec] Delta v applied during one speed control loop cycle
float vMax = 20; // [cm/sec] Max speed of left/right motor
float cmPerHalfPeriod = 1; // [cm] Distance traveled by motor per half period (duration of true or false)

// =====
region SetSpeeds:

initial state SetSpeeds "" {
// Possible syntax, to replace the clk flag and the GenClk region,
// and not having to condition delayed triggers with "clk &":
// clock type = soft, min = pSetSpeedsMinUsec, max = pSetSpeedsMaxUsec
bool clk; // Local clock

// =====
region ProcessInputs:

initial state Init
--> Running immediate;

state Running {
entry / v = 0;

// =====
region CalcV:

initial state Pause
--> Accel with clk & accel & !decel
--> Decel with clk & decel & !accel;

state Accel
--> CheckMax immediate with / v += dV;

state Decel
--> CheckMin immediate with / v -= dV;

state CheckMax
--> SetPeriod immediate with v <= vMax
--> SetPeriod immediate with / v = vMax;

state CheckMin
--> SetPeriod immediate with v >= -vMax
--> SetPeriod immediate with / v = -vMax;

state SetPeriod
--> Pause immediate with v == 0 / pMotorUsec = 0
--> Pause immediate with / pMotorUsec =
'(int) (1000000 * cmPerHalfPeriod / v);
}
o-> Running with clk & stop / pMotorUsec = 0;

```

```

// =====
region GenClk:
initial state GenClkState {
int myWakeMinUsec, myWakeMaxUsec;

initial state Init
--> AssertWakeTime immediate with / clk = true;
myWakeMinUsec = currentUsec + pSetSpeedsMinUsec;
myWakeMaxUsec = currentUsec + pSetSpeedsMaxUsec;

connector state AssertWakeTime
--> Pause immediate with / wakeUsec = myWakeMaxUsec; // Initialize wakeUsec
@layout [layerConstraint] LAST
state Pause
--> AssertWakeTime with currentUsec < myWakeMinUsec / clk = false
--> Init;
};

// =====
region CtrlMotor:

initial state CtrlMotor "" {
// Possible syntax, to replace the clk flag and the GenClk region,
// and not having to condition delayed triggers with "clk &":
// clock type = hard, min = 1000000 * cmPerHalfPeriod / vMax // = 50000
bool c; // Local clock

// =====
region GenClk:

initial state GenClkState "" {
int myWakeUsec;

initial state Stopped
--> AssertWakeTime immediate with pMotorUsec > 0 / myWakeUsec = currentUsec +
pMotorUsec; clk = true;

connector state AssertWakeTime
--> Running immediate with / wakeUsec min= myWakeUsec; // Initialize wakeUsec

@layout [layerConstraint] LAST
state Running
--> ResetClock with / clk = false;

connector state ResetClock
--> AssertWakeTime immediate with pMotorUsec > 0 & currentUsec < myWakeUsec
--> Stopped immediate;
};

// =====
region Motor:

initial state Low
--> High with clk / motor = true;

state High
--> Low with clk / motor = false;
};

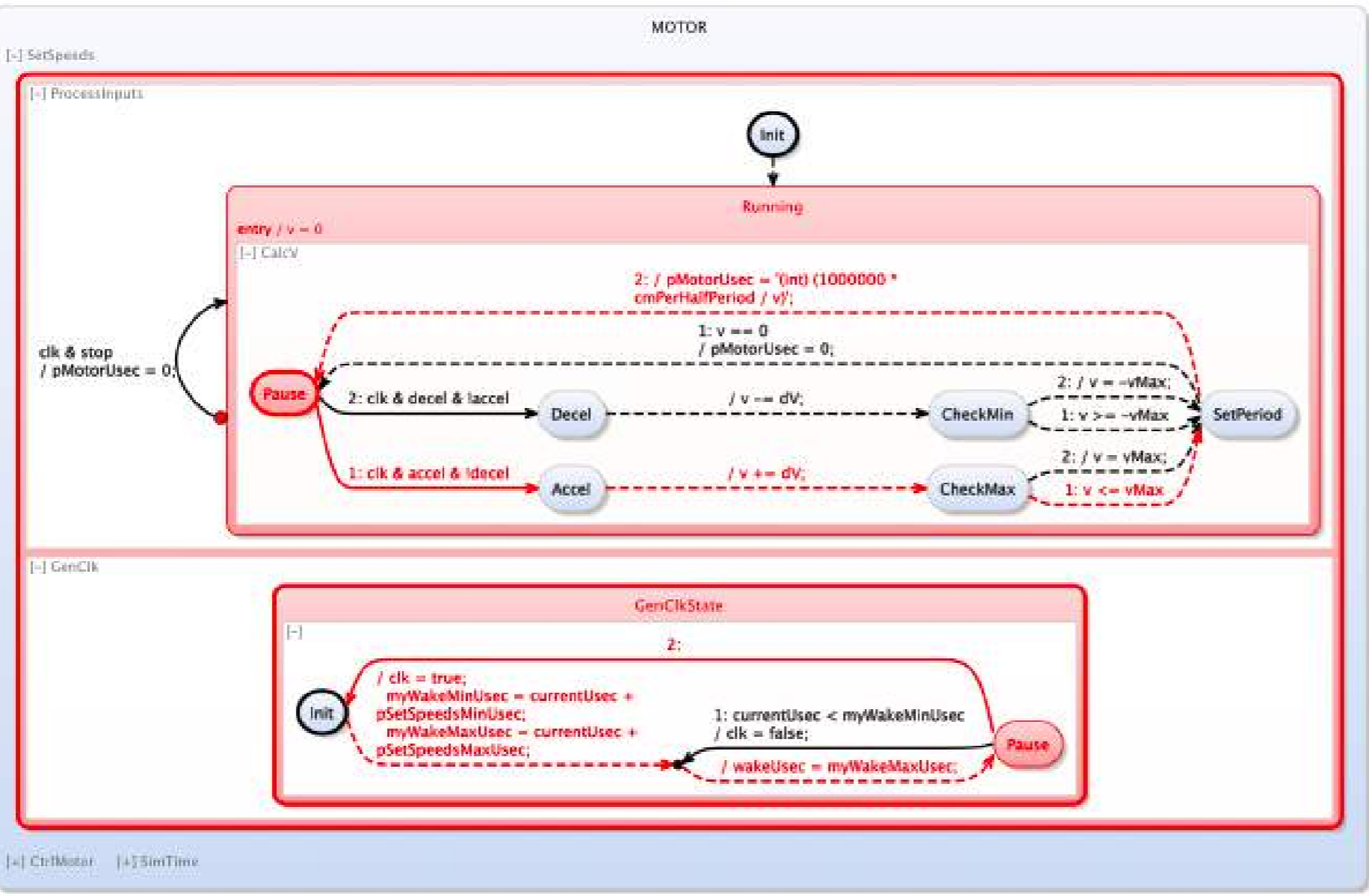
// =====
region SimTime:
initial state SimTimeState "" {
during / pUsec = currentUsec;
currentUsec = pre(wakeUsec);
pUsec = currentUsec - pUsec;
pMinUsec min= pUsec;
};
}
}

```

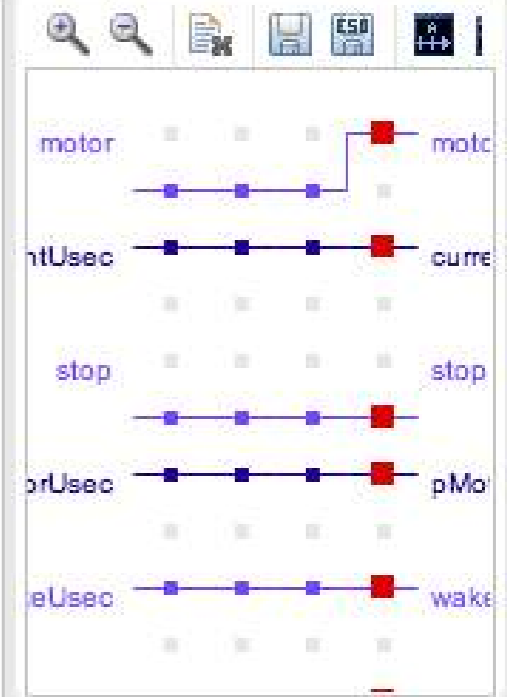
Now use **KIELER** to  
 synthesize **graphical**  
**SCChart** with **ELK**  
 and simulate ...



Diagram



Synchron

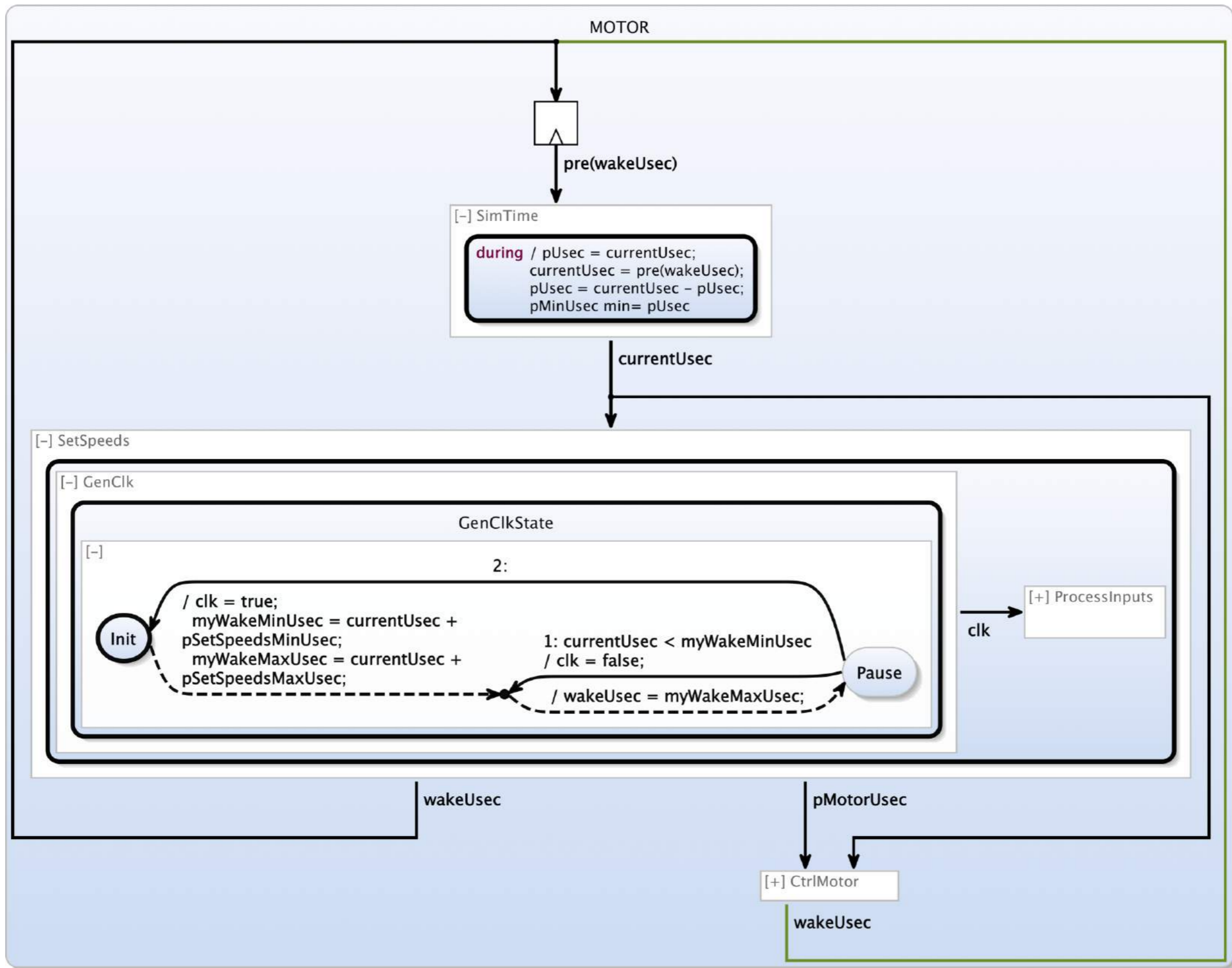


Data Tab

[4]

P	Key
<input checked="" type="checkbox"/>	*accel
<input checked="" type="checkbox"/>	currentUsec
<input type="checkbox"/>	decel
<input checked="" type="checkbox"/>	motor
<input checked="" type="checkbox"/>	pMinUsec
<input checked="" type="checkbox"/>	pMotorUsec
<input checked="" type="checkbox"/>	pUsec
<input type="checkbox"/>	state
<input type="checkbox"/>	stop
<input type="checkbox"/>	transition
<input checked="" type="checkbox"/>	v
<input checked="" type="checkbox"/>	wakeUsec

KIEM Execution



[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden

*Augmenting State Models with Data Flow*

Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]